SEMANTIC INTERPRETATION OF AN ARTIFICIAL NEURAL NETWORK

THESIS

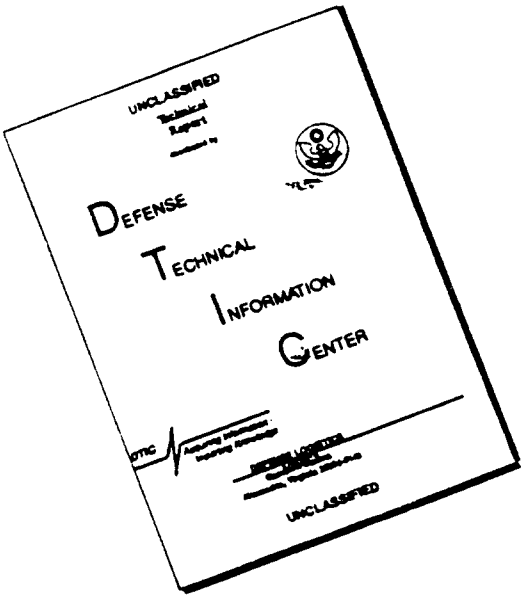
Stanley Dale Kinderknecht
Captain, USAF

AFIT/GCS/ENG/95D-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GCS/ENG/95D-07

SEMANTIC INTERPRETATION OF AN ARTIFICIAL NEURAL NETWORK

THESIS

Stanley Dale Kinderknecht
Captain, USAF

AFIT/GCS/ENG/95D-07

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/GCS/ENG/95D-07

SEMANTIC INTERPRETATION OF AN ARTIFICIAL NEURAL NETWORK

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Stanley Dale Kinderknecht, B.S.

Captain, USAF

December 1995

Acknowledgements

This paper reflects long, arduous hours of research and writing. Although not always enjoyable, breaking ground on something that had never been accomplished before proved exhilarating. I could not have reached this far nor achieved this much without intimate guidance and supervision provided by my committee.

I would like to thank my advisor, Dr Steve Rogers, for his strong commitment to the goals for which I strove. For unwavering belief in me and my potential to succeed, I would like to thank my readers, Dr Eugene Santos, Jr. and Maj Gregg Gunsch. Without their meticulous perusal of this developing document—as well as their constant words of encouragement—achieving my research goals and producing a truly exemplary document would not have been possible.

Most emphatically, I sincerely wish to thank my wife, Lisa, whose patience and understanding proved infinite. Finally, I want to thank my six children, Christopher, Corwin, Candace, Cian, Collin, and Cameron, for their involuntary acceptance of my pursuit of a Master's degree. Their calls for "Dad" went unanswered altogether too often. After a year and a half, I gladly focus my thoughts and attention back to them, my loving family.

Stanley Dale Kinderknecht

Table of Contents

List of Figures

List of Tables

Abstract

Recent advances in machine learning theory have opened the door for applications to many difficult problem domains. One area that has achieved great success for stock market analysis/prediction is artificial neural networks. However, knowledge embedded in the neural network is not easily translated into symbolic form. Recent research, exploring the viability of merging artificial neural networks with traditional rule-based expert systems, has achieved limited success. In particular, extracting production (IF...THEN) rules from a trained neural net based on connection weights provides a valid set of rules only when neuron outputs are close to 0 or 1 (e.g. the output sigmoid function is saturated). This thesis presents two new ways to interpret neural network knowledge. The first, called Knowledge Math, extends the use of connection weights, generating rules for general (i.e. non-binary) input and output values. The second method, based on decision boundaries, utilizes the inherent border between output classification regions to draw symbolic interpretation. The Decision Boundary method generates more complex symbolic rules than Knowledge Math, but provides valid feature relationships in the uncertain regions around the midpoints of the neuron output functions. The main result is a complementary relationship between Knowledge Math and Decision Boundaries, as well as subsymbolic and symbolic knowledge representations for a general multi-layer perceptron.

SEMANTIC INTERPRETATION OF AN ARTIFICIAL NEURAL NETWORK

I. Introduction

Recent advances in machine learning have opened the door for applications to many difficult problem domains. In particular, artificial neural networks have achieved great success for stock market analysis and prediction. However, neural networks lack an explanation facility that make traditional expert systems more desirable in these tough domains. It would be nice to somehow merge these two contrasting approaches, achieving the benefits of both.

This paper addresses the problem of augmenting a neural network with an expert system explanation capability. First, the strengths and weaknesses of each machine learning technique must be well understood. Although seemingly opposed, artificial neural networks and expert systems are fundamentally two successful applications of machine intelligence.

1.1 What is Intelligence?

Winston defines *artificial* intelligence (AI) as the study of the computations that make it possible to perceive, reason, and act [23]. The ultimate AI goal, then, is to make a machine think and act like a human. However, the mechanisms with which humans (or any other species) perform these cognitive tasks are not well understood. On the one hand, the human brain contains a vast, interconnected network of neurons. Although the simplified physical properties of these brain cells can be modeled[16], precisely how they account for *thought*, *memory*, or *intelligence* remains a mystery. On the other hand, the mind is a nebulous inferencing device, though its operation is grounded in well-ordered logic and rules. Nobody knows exactly how one can account for the other.

With their limited understanding of human cognition, researchers have found this lofty goal—construction of an artificial person—difficult (if not impossible) to achieve. Still, their dream has elicited a huge amount of research in a broad range of domains, including

robotics, vision, natural language recognition, and reasoning. While the quintessential goal still seems largely untenable, emergent technologies, such as *expert systems*, are beginning to enjoy commercial and academic success.

1.2 Expert Systems

The human brain stores all kinds of knowledge, from the most trivial bits of information to abstract concepts and complex ideas, in an easily accessible manner. The quest for a tool which makes the representation of a large amount of knowledge possible, as well as consistent and effectively usable, is one of the basic problems of artificial intelligence[22]. Expert systems, computer systems that emulate human decision-making, represent very successful approximate solutions to the classic AI problem of programming intelligence[7]. Basically, there are two main approaches to expert system design based on two distinct classifications of knowledge representation: *symbolic* and *subsymbolic*.

1.2.1 Rule-Based Expert Systems. A conventional expert system maintains knowledge *explicitly* as a set of human-understandable *facts* and *rules* (i.e. as *symbolic* knowledge). Such an expert system consists of three essential parts: a rule base, a working memory, and an inference engine. These are roughly analogous to a human's long-term memory, short-term memory, and cognitive processing, respectively. Rules reside in the rule base. Facts entered by the user or inferred by the system exist in the working memory. The inference engine matches rule antecedents within the rule base against facts in the working memory. A match causes a rule to fire; the rule's consequent represents an action to be performed and/or a new fact to be inserted into the working memory.

A *rule-based* expert system inferences over a set of production (**IF ... THEN**) rules. Production rules represent a "natural" encoding of human knowledge; much of human problem solving can be expressed in this form[15]. In fact, **IF ... THEN** rules simplify knowledge acquisition. Experts generally express their experiences in a way that translates readily into production rules. For example, an expert's statement

When it rains, the sidewalk gets wet.

can be directly encoded as the production rule

IF *it is raining*, **THEN** *the sidewalk is wet.*

Due to the explicitness of the knowledge, the system works well with inaccurate and incomplete information, such as when employing confidence factors. Such systems are relatively easy to create, and validation of the knowledge base is straightforward.

1.2.2 Artificial Neural Networks. Artificial neural networks (or, simply, neural nets) represent an AI computational paradigm radically different from rule-based expert systems. Like its biological counterpart, an artificial neural net is composed of independent processing elements called *neurons.* Unlike a rule-based expert system, a neural net does not represent a piece of knowledge, such as a fact or a rule, as an atomic, isolated entity (i.e. an element in a database, or an address in memory). Instead, knowledge is distributed mathematically across the network in the form of weighted connections between neurons. Thus, the neural net's knowledge is not inherently discernible; rather, it is *implicit*, or *subsymbolic.* Generally, identifying an individual neuron's role in a network is difficult.

The neural net knowledge base is created automatically by a learning algorithm. Neural nets implicitly discover fundamental relationships (rules) embedded in a set of training data. Learning procedures can be derived and their properties mathematically analyzed because in these systems, knowledge representations are extremely impoverished [17]. Working with incomplete information, and providing justification of inference, is limited or even impossible. Validation of completed system is dependent on statistical analysis and performance.

While both approaches have enjoyed notable success in a number of problem domains[1], limitations inhibit either from becoming the general AI tool researchers desire. However, by mapping a new problem domain to one of these opposing strategies, one could achieve a reasonable implementation. Which, then, should be chosen?

1.2.3 Mind vs. Brain. Traditional expert systems and neural nets seem to be diametrically opposed in their functionality. Neural nets attempt to model physical attributes of the brain; expert systems model the abstract notion we call intelligence. That is, a neural net can be viewed as our organic computer's *hardware*; an expert system represents our biological *software*. While this analogy suggests some implicit connection, our meager silicon implementations remain disjoint. Nowhere is the contrast between the symbolic and subsymbolic paradigms more dramatic than in learning.

Expert systems, built through knowledge extraction and rule-based development, impose rigid descriptions of facts and rules. An articulate human expert must extract regularities from his experience and express them in the comprehensible, explicit form of rules. Learning a new concept entails creating something like a new schema. Because schemata are such large and complex knowledge structures, developing automatic procedures for generating them in original and flexible ways is extremely difficult[17]. Large systems require careful design and can be unwieldy and difficult to maintain if not carefully developed and designed[22]. However, they work well on ordinary digital computers. The average development time is 12 to 18 months [1][22].

Neural nets, built through training with data examples, require many input vectors whose expected outcome are known. On the other hand, no expert is needed. Neural nets *learn*, through well-defined learning algorithms, to classify items based on input features.

[1]Successful application areas of these machine intelligence paradigms include:

- Traditional (i.e. rule-based) expert systems have achieved significant success in such diverse areas as configuration and control, monitoring/diagnosis, tutoring/instruction, and planning[7].
- Neural nets are particularly useful in any application that can be mapped to a pattern matching activity (i.e. drawing conclusions from a set of input "features"). Neural nets have been applied successfully to target recognition, radar signal processing, and text-to-speech conversion, and hold great potential for vision, voice recognition, on-line system diagnosis, mission planning, and optimization problems[16].

Neural nets implicitly discover essential rules embedded in the training data. A new schema comes into being gradually, as the strengths of neurons slowly shift in response to environmental observation, and new groups of coherent neurons slowly gain important influence in the processing[17]. Large systems need long-term, computationally exacting learning, with uncertain results[22]. For all but the smallest networks, the best performance comes from the use of accelerator-assisted or specialized parallel chip boards[1]. Development time is as little as a few weeks or months, and additional learning is possible[1][22].

Drawing a conclusion in a rule-based system consists of matching facts in its working memory against rules in the rule base. A firing rule often injects new information into the working memory, causing a new *state of the world*; for every change in the state of the world, the *exhaustive search* through the rule space must begin anew. Alternatively, a neural net performs, for any arbitrary input vector, a constant number of mathematical calculations to draw its conclusion. That is, a decision is always found in constant time $(O(1))$[2]. Thus, as the size of the rule-base increases, an equivalent neural net will be faster (especially if implemented in hardware).

Is it possible to combine these ostensibly disjoint AI techniques—rule-based expert systems and neural networks—to solve the stock market prediction problem efficiently and effectively? In particular, can this hybrid *expert network* realize the capabilities of both types of systems while minimizing the deficiencies of either?

1.2.4 Hybrid Systems—Expert Networks. Rather than contradictory, the advantages and disadvantages of rule-based and neural approaches to expert system design, like the human brain and mind, are in fact complementary[22]. It seems natural to conflate these two concepts, integrating advantages of both. Caudill suggested some general models of such hybrid systems, shown in Figure 1.1[1].

The most appropriate model to use depends on the nature of the problem domain. Rule-based expert systems are good when an expert is available, articulate, and explicitly cognizant of his decision processes. Neural nets are good when the expert does not know for

[2]This is true for a static neural net. Especially during training, the net architecture may be dynamic—nodes may be added or removed. In this case, the execution time depends on the number of neurons, n; the net operates in linear time $(O(n))$.

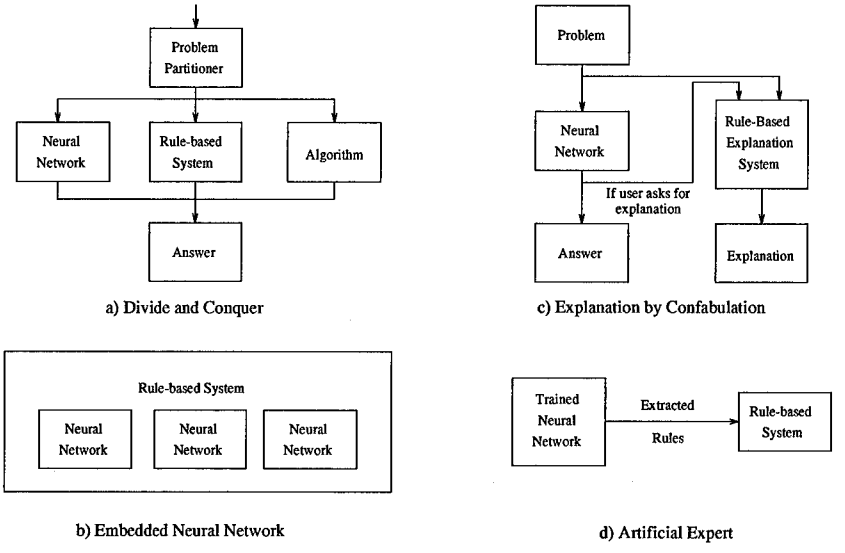


Figure 1.1 **Expert Network Architectures.** Four basic models of combining neural nets with traditional expert systems, including: **(a) Divide and Conquer.** Partition the problem into subtasks and apply the most appropriate method to each; **(b) Embedded Neural Network.** As noted in Section 1.2.2, neural nets perform pattern matching extremely well. Matching the state of the world to rule antecedents (**IF** clauses) is inherently a pattern matching activity. Use the net within the rule base; **(c) Explanation by Confabulation.** Augment a normal neural net with a small rule base for a limited explanation (or, more precisely, a rationalization) facility; and **(d) Artificial Expert.** A trained neural net acts as an expert; extract semantic rules from the subsymbolic net to populate a rule-base.

certain which features and relationships are crucial, which are redundant and superfluous, and what knowledge is actually contrary.

Consider a stock market predictor. Analysts (more or less) successfully predict the market every day based on certain sets of indicators, such as short-term and long-term trends. Some of these analysts are truly expert, in that they predict correctly significantly more often than they are mistaken. However, even these experts can readily express neither the importance of individual indicators (features), nor the relationships between them. An expert system that could learn these fundamental relationships would be a great benefit, suggesting a strong mapping of this domain to artificial neural networks.

However, neural nets provide little or no explanation capability. The ability to explain itself is crucial in an expert system. Humans typically fear and distrust computers.

More importantly, if a neural net cannot explain itself, then the important features and relationships (from the computer's point of view) cannot be ascertained. Hence, the net will not help to increase domain knowledge.

Due to the explicit nature of the knowledge, rule-based systems have perfect explanation abilities. Is it not possible to provide a rule base-like explanation facility while taking advantage of a neural net's learning potential?

1.3 Problem Statement

The purpose of this research is to elicit rules from an artificial neural network. In particular, a trained neural net will be augmented with an explanation facility. When a user applies an input vector, the net will not only provide an output classification but in response to user prompting, will supply salient input relationships in the form of rules as well.

1.4 Scope of Research

This research presents a new method of interpreting the subsymbolic net knowledge, based on decision boundaries. A standard multi-layer perceptron will be used as a basis for symbolic interpretation. The net will be trained normally over a variety of illustrative problem domain, and the trained net will be augmented with an ability to explain the output classification in symbolic form.

1.5 Thesis Organization

The following chapter describes much of the current research into combining neural nets and expert systems. Fu's method of translating the neural net into a set of symbolic rules is explored more deeply.

In Chapter IV, a method called Knowledge Math is explored. Knowledge Math extends Fu's concepts, utilizing not only connection weights, but also the input values, to establish a means to *interpret* rather than *translate* an artificial neural network. In particular, the neuron input and output values are not restricted to binary representations.

As a result, symbolic rule generation is possible for individual input vectors in certain regions of the input space.

Then, Chapter V describes the Decision Boundary method. This method provides a symbolic interpretation ability for the regions near the midpoints of the neuron output functions, which remain ambiguous in the Knowledge Math method.

Finally, conclusions and recommendations are summarized in Chapter VI.

II. Related Research

Hybrid models proposed in recent literature are invariably compromises, either preferring explanation abilities and work with incomplete information to the detriment of an automatic adaptability, or preferring a neural network architecture that sacrifices explicit work with information.

What kind of model provides the greatest potential for meeting the needs of an artificial stock market expert network?

2.1 Common Hybrid—Net→Rule Base

Of the four hybrid approaches presented by Caudill[1] (see Section 1.2.4), the Artificial Expert offers the advantage of rule-based inferencing and explanation capabilities, while capitalizing upon an artificial neural network's ability to train without an expert. In fact, many researchers have recently exploited this concept. In such a hybrid system, a set of production rules is extracted from a trained neural net. This rule set, which may need to be manually refined or extended, forms the basis of a traditional rule-based system.

According to Caudill, the rule-extraction process is tedious, and the results may not be quite what one might expect[1], but it is possible to perform. However, an intuitive symbolic representation of subsymbolic knowledge is not obvious. As mentioned in Section 1.2.2, one cannot look to the role of individual neurons in identifying an inference chain. How, then, are researchers able to extract semantic rules from a neural net? By and large, researchers build into their nets the extraction capabilities. In other words, their systems employ special net architectures.

2.2 Special Architectures

One problem in integrating both neural and rule-based approaches is the creation of the knowledge base when only some rules and the example data are available. McMillan, Mozer, and Smolensky[13], Samad[18], and Yang and Bhargava[24] proposed heuristics to construct the neural knowledge base from explicit rules. Similarly, Frasconi et al[3], Handelman, Lane, and Gelfand[8], and Hruska, Kuncicky, and Lacher[9] considered hybrid

learning, when neural network weights are initialized from explicit rules (e.g. using linear programming), and learning from examples is then viewed as a refinement process to cover uncertainty.

Rather than providing an explanation facility to a trained neural net, however, this work addresses the viability of predetermining the initial neural net structure from a rule base. A net constructed in this manner may better serve the rule-extraction process. However, the ease of interpretability is bought with a decrease of generalization ability.

2.3 Examples

Towell, Shavlik and Noordewier[21] claimed a knowledge- based neural network could outperform a standard backpropogation network, as well as other related learning algorithms (including symbolic and numerical ones) in certain domains, suggesting a means of increasing domain knowledge. Domain knowledge, in the form of Boolean rules, was initially incorporated into a neural net. After training, refined rules were extracted from the network. However, only knowledge about what had changed was made available; it was still impossible to discern why certain changes occurred.

Similarly, Yeung and Fong proposed a rule-mapped neural network model, incorporating domain knowledge initially[26]. They later advocated a technique to trigger modifications on the symbolic knowledge base using network performance[25]. They presented a tool, called a knowledge matrix, to symbolically interpret the network's response to an input. Consequently, they maintain learning is made more controllable and comprehensible.

Gallant[6] suggested a means of generating a knowledge base from a trained neural net, and an "inference engine" to interpret the knowledge base. His method uses a simple neural net model with better interpretation of knowledge representation but with a weaker learning algorithm.

Fu[4] recognized a bidirectional linkage between the net and the rule base. In his model, useful domain attributes and concepts were first identified and linked in a way consistent with initial domain knowledge, and then the links were weighted properly so as to maintain the semantics. In this manner, Fu attested, a rule-based system could be

mapped into the neural net, and the net's knowledge could be transferred back into a rule-base. However, this last transformation was not obvious. In [5], Fu supplied an algorithm to perform this latter translation; rules were formed based on searching through all possible combinations of neuron input weights. He recommended several pruning heuristics to make this exhaustive search feasible.

Kane and Milgram[10] explored logical rule extraction by forcing certain neurons to realize elementary logical operations (i.e. AND, OR, or NOT). They constrained connection weights to a finite set of values and forced all neurons to operate in the saturated part of their output function (i.e. for the sigmoidal neuron, the output was arbitrarily close to a binary **0** or **1**). A truth table generated for each neuron indicated what the saturated output should be, based on combinations of inputs, to achieve the chosen logical operation. Associated rules were directly observed from the truth table.

Both Fu and Kane constrained the neuron outputs to 0 or 1, even though the actual output of an activation function (e.g. the sigmoid function) is a real value in the interval [0,1]. The hard-limiting 1 or 0 activation forces the associated rule to either fire or not fire. However, in a rule-based system it is often desirable to associate uncertainty with the rules. That is, based on premise strengths, the consequent may be true with a certain probability (certainty factor). The real-valued input weights provide a natural means of describing this uncertainty.

Fuzzy logic deals with the uncertainty of verbal expression[27]. Fuzzy rules are production rules whose certainty factors are classified into broad categories. For instance, a rule consequent may be true with a HIGH, MEDIUM, or LOW likelihood. According to Zadeh, fuzzy logic is an excellent means to combine artificial intelligence methods[28].

Mitra and Pal[14] described an expert network model based on a fuzzy multi-layer perceptron. Their system inferred the output class membership value (e.g. HIGH, MEDIUM, or LOW) and generated a measure of certainty expressing confidence in the decision. In the case of partial inputs, this model was capable of querying the user for the *more important* input feature information needed. The inferencing procedure utilized connection weight

magnitudes. The trained neural net constituted a knowledge-base for the given application domain, from which fuzzy production rules were extracted.

Tazaki and Inoue[20] implemented an architecture enabling automated extraction of fuzzy rules using a multi-layer perceptron with a planar lattice structure on the hidden layer. Each lattice neuron in the hidden layer was assigned a fuzzy proposition composing a fuzzy rule. The network learned structurally with generation/annihilation of neurons. As a result, fuzzy rules could be extracted.

2.4 Why this method?

Special architectures restrict the applicability and generality of the neural net and, thus, the expert system as a whole. In particular, discarding the neural net in favor of the resulting rule base ignores the benefits of the net, such as computational speed[1]. Furthermore, the rule base is at best an approximation of the precise knowledge distributed across the net. Researchers supporting the Artificial Expert paradigm have suggested the extracted knowledge base performed as well as, or even better than, the original neural net[21] [6][5]. They view the symbolic rules, not as an approximation, but as a generalization of the neural net knowledge. However, given the neural net's goal of generalization to the training data, this result seems unlikely in a general net.

Is it not possible to train a general (even a generic) neural net to learn a problem domain (say, stock market prediction), and, once trained, use the neural net with some measure of explanation capability?

The following chapters present the evolution of a technique to extract symbolic knowledge from a trained neural net. Unlike this previous work, the goal of this research is to avoid the need to form a stand-alone rule base or artificially constrain the net architecture. From a general net, symbolic rule *relationships* are discovered.

A unique approach to soliciting symbolic information from a subsymbolic net employing the concept of *decision boundaries* is presented in Chapter V. First, however, the

[1] A large rule base is slow and difficult to maintain. A neural net (particularly implemented in parallel hardware) takes a constant number of calculations (i.e. constant time within a static net, linear time with respect to the number of neurons in a dynamic architecture) to find an output for a given input vector.

special architecture approach is extended in an attempt to interpret net knowledge symbolically for an arbitrary input vector. Chapter III examines Fu's rule generation technique more closely.

Although Fu employs a special architecture and forms an independent rule base, his methodology is founded on some of the basic neuron mathematical properties. That is, he coaxes semantic meaning from actual knowledge distributed throughout the net. This seems a good place to begin an attempt to provide an explanation facility to the net.

III. A Closer Look at Fu

Fu[4] describes an intelligent hybrid system that combines the two fundamental Artificial Expert notions: injecting symbolic knowledge into the net and eliciting meaningful rules from the net. Established production rules determine initial neural net connections and weights. Domain attributes and concepts are identified and linked in a way consistent with initial domain knowledge, and the links are weighted properly to maintain the semantics. Hidden units and additional connections are introduced as appropriate. This primitive structure evolves through self-adaptation. Finally, the trained neural net, a refinement of the initial knowledge base, is translated back into rules.

The rule-based inference system is mapped into the neural net in the following manner. Data attributes or variables are assigned input units, intermediate concepts or hypotheses are assigned hidden units, and target concepts or final hypotheses are assigned output units. Hidden and output neurons represent conjunctive units. All initial rules must be rewritten into the form

$$\mathbf{IF}\alpha_1 \text{and} \alpha_2 \text{and} \alpha_3 \mathbf{THEN} \beta \tag{3.1}$$

This results in a set of simplified rules, each having an antecedent (premise) consisting of one or more conditions, and a single consequent. The premise is assigned a hidden neuron. Each condition corresponds to an assigned attribute or concept (input node). The consequent corresponds to an output concept node. If the premise consists of a single condition, the corresponding input node can be connected directly to the output node representing the rule's consequent; no hidden node is needed.

During training, the rule strengths are adjusted via two rounds of back propagation. Between rounds, hidden units are clustered based on similarity of their input weight vectors. When the network performance gets stuck, new hidden units are added to generate new concepts and rules.

Finally, the revised neural net is translated into rules. The translation algorithm heuristically searches the rule space distinguishing between positive and negative attributes.

Positive attributes link to the concept with positive weights, and negative attributes link with negative weights. Confirming rules, with an asserted conclusion, C, are generated by exploring combinations of positive attributes in the presence of various negated attributes. Similarly, disconfirming rules, with a negated conclusion, $\neg C$, account for combinations of negated attributes with various positive attributes.

Initial knowledge strength affects the model design for revising the knowledge. If there is no initial knowledge at all, the net degenerates into a typical, randomly weighted, fully-connected multi-layer perceptron. If the initial knowledge is strong, it can be mapped into the net without adding any hidden units or connections. As a result, no additional rules will be extracted.

Fu's translation approach accounts for the distributed nature of the net knowledge. Of the weighted connections, the *weights* provide key information; the neuron inputs are binary. How can the input weights alone lead to symbolic rules?

3.1 Implementing Fu as a Net Interpreter

Fu suggests that a rule generated from a neural net has the production form

IF the premise, **THEN** the conclusion. (3.2)

Specifically, the combination of weights at each neuron, along with the neuron output, realizes a production rule. The neuron output, y, is a function of the weighted inputs:

$$y = f(w_1 * x_1 + w_2 * x_2 + \cdots + w_n * x_n + \theta). \tag{3.3}$$

Fu exploits the notion of a neuron *firing.* That is, when the neuron's activation function is greater than 0, the output is a binary **1**; otherwise, it is a **0**.

The binary nature of Fu's net is essential, since he considers only combinations of weights to form his rules. That is, he assumes a given weight w_i, associated with neuron input x_i, either fully contributes or fully fails to contribute to the neuron output (and the resulting rule). This discrete ON/OFF property of the weights suggests binary

input values; otherwise, his rules would be increasingly more inaccurate as the activation function approached 0. In the latter case, an input that should contribute only a little (either positively or negatively) to the neuron's firing potential would in fact be assumed to contribute as much as possible or not at all.

While this binary functionality can be achieved by a hard limiter non-linearity for each neuron, hard limiters have two serious disadvantages:

- hard-limited nets are notoriously hard to train, and
- strict binary functionality restricts the general utility of the net.

The sigmoid can be forced to perform a quasi-binary role (as noted in Appendix **??**). By supplying a large "steepness" coefficient, λ, one can virtually guarantee the neuron output, y, will always be "close" to **1** or **0**. As the steepness increases, however, the sigmoid more closely approximates a hard limiter, with the same drawbacks. The trick is to find some λ that will allow good training while providing a strong enough binary characteristic[1].

Assuming binary inputs, the output must also be binary by the following argument. A neuron input must come from either (1) a network input, or (2) the output of another neuron. It follows naturally that to ensure all inputs are binary, all outputs must also be binary[2]. Furthermore, each input weight, w_k, either contributes or fails to contribute to the output, based on whether x_i is a **1** or a **0**. Ordering the inputs such that all **1** inputs are listed first (i.e. $\forall_{k,l}[x_k = 1 \wedge x_l = 0] \rightarrow x_k$ before x_l), the neuron equation becomes

$$y = w_1 * (1) + \cdots + w_k * (1) + w_{k+1} * (0) + \cdots + w_{k+l} * (0) + \theta \quad k + l = n \qquad (3.4)$$

Each contributing weight can be either positive or negative. Positive weights, w^+, increase the likelihood the neuron will "fire" (its output will be a **1**), while negative weights,

[1]It is virtually impossible to realize a true ON/OFF system. For example, IC chip voltage levels range roughly between 0 an 1 volt, with two distinctive cut-off thresholds defined (e.g. TTL ON/OFF thresholds of 0.3 and 0.7 V) to "unambiguously" delineate a **1** representation from that of a **0**. Theoretically, the null-area in between covers an area where the transistor will essentially never operate. A similar argument can be applied to neuron output if the sigmoid is steep enough.

[2]This is not strictly true in the output layer, depending on the method of output classification. For the time being, assume the output class is binary encoded (e.g. 3 output neurons can represent up to $2^3 = 8$ output classes). Thus, even network outputs have a binary functionality.

w^-, decrease this potential. Finding a neuron rule consists of searching for some combination of positive and negative weights such that the total contributing weight (positive contributors minus negative contributors) exceeds the neuron threshold θ, ensuring the neuron fires[3]. Letting w_i represent a contributing weight and w_j represent a non-contributor, Equation 3.4 becomes

$$y = w_1^+ + \cdots + w_i^+ + w_1^- + \cdots + w_j^- + \theta \quad i + j = k. \tag{3.5}$$

A neuron rule consists of a series of antecedents, each reflecting a contributing or non-contributing weight. That is, contributing weight w_i corresponds to an asserted antecedent A_i, while the non-contributor, w_j, corresponds to a negated antecedent, A_j. Mapping weights to antecedents ($\{w\} \rightarrow \{A\}$) and the output y to the consequent C, a specific neuron rule can be expressed as

$$\textbf{IF} A_1^+ + \ldots + A_i^+ + \neg A_1^- + \ldots + \neg A_j^- \textbf{ THEN } C \text{ (or } \neg C). \tag{3.6}$$

Finding the neural net rules consists of collapsing the set of neuron rules to relate input antecedents (antecedents associated with *net* inputs) to output consequents (conclusions associated with *net* outputs). In this light, hidden nodes can be considered intermediate conclusions; however, they contain no semantic value because of the distributed nature of the knowledge base, and they must be removed. To collapse neuron rules[4]:

1. FIND A RULE CONTAINING A HIDDEN NODE ANTECEDENT, A_{H_i}. IF NO SUCH RULE EXISTS, THEN DONE.
2. FIND ALL RULES WHOSE CONSEQUENT REFERENCES THE HIDDEN NODE (C_{H_i}).
3. FOR EACH RULE FROM (2), GENERATE A NEW NET RULE REPLACING A_{H_i} IN (1) WITH THE ANTECEDENTS IN (2).
4. GO TO (1).

[3] A neuron output of **1** results in a rule with an asserted consequent, called a *confirm* rule ($[y = 1] \rightarrow C$). Conversely, a **0** output indicates a negated consequent in a *disconfirm* rule ($[y = 0] \rightarrow \neg C$).

[4] For sake of clarity, let the antecedent/consequent subscript indicate the input/output. For instance, A_{I1} indicates a weight associated with the first input, while A_{H3} represents a weight associated with an input from the third hidden node. Note that the consequent indicates the neuron associated with the rule, such as C_{H2} or C_{O1}. Although this restricts the MLP architecture to a single hidden layer, Cybenko showed that this is sufficient for most real-world problems[2]; at the very least, it will suffice for purposes of illustration.

Note that a hidden node antecedent must match a hidden node consequent *exactly.* That is, an asserted antecedent A_{H_i} will match only to a confirming rule with the positive consequent C_{H_i}. Similarly, a negated antecedent $\neg A_{H_i}$ matches only a disconfirming rule with negative consequent C_{H_i}. Thus the neuron rules

R1: $A_{I1} \wedge \neg A_{I2} \rightarrow C_{H1}$
R2: $A_{I3} \rightarrow C_{H1}$
R3: $A_{I4} \wedge \neg A_{I1} \rightarrow C_{H2}$
R4: $A_{I5} \wedge \neg A_{I4} \rightarrow \neg C_{H2}$
R5: $A_{H1} \wedge \neg A_{H2} \rightarrow C_{O1}$

become the neural net rules

R1: $A_{I1} \wedge A_{I5} \wedge \neg A_{I2} \wedge \neg A_{I4} \rightarrow C_{O1}$
R2: $A_{I3} \wedge A_{I5} \wedge \neg A_{I4} \rightarrow C_{O1}$

Disconfirming **neuron** rule R4 contains a negated consequent $\neg C_{H2}$ that is not matched to any antecedant in the neuron rule set. Hence, it is simply ingored.

3.2 A Modicum of Modification

How, then, could this methodology be used to supply an explanation capability to a working neural net? Implementing Fu's system completely, one can visualize the creation of a neural net explainer, as in the Explanation by Confabulation model (see Section 1.2.4). The net-derived rules form the basis of a small, separate rule base, such that the neural net is used in normal day-to-day activity. When the user wants an explanation, the rule-base is tasked to draw a likely explanation.

However, rationalization may be unacceptable to the user. A better solution is to construct these neuron rules, not as a set of all possible rules that must be collapsed, but as a single input to output "chain" representing a unique rule for a particular input vector. For binary net inputs, there exists a finite number of unique input vectors (2^m, where m is the number of inputs). Another way of looking at it is that there are as many unique rules (an upper bound anyway) as there are "bins" in the binarized input space.

Consider the neural net in Figure 3.1. If the inputs can be binarized, a single rule can be constructed in this manner:

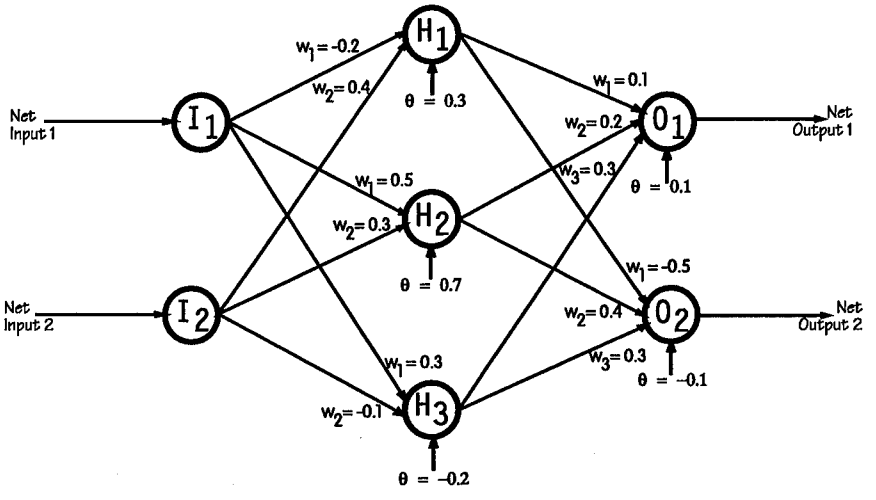


Figure 3.1 **Multi-Layer Perceptron.** With the neuron bias, θ, and the weight, w_i, associated with each neuron input, x_i.

1. FOR EACH HIDDEN NODE, CONSTRUCT THE *single* NEURON RULE BASED ON BINARY NET INPUTS. FOR EXAMPLE, HIDDEN NODE H_1 IS RELATED TO THE INPUTS $I_1 = 1, I_2 = 0$ BY THE RULE $A_{I1} \wedge \neg A_{I2} \rightarrow C_{H1}$.
2. DETERMINE EACH HIDDEN NODE OUTPUT AND, THUS, WHETHER THE NEURON RULE IS *confirming* OR *disconfirming*. FOR THIS EXAMPLE, ONLY INPUT I_1 CONTRIBUTES. SINCE $w_{I1} = -0.2$ IS LESS THAN THE THRESHOLD $\theta_{H1} = 0.3$, THE OUTPUT IS A **0** (THE RULE IS DISCONFIRMING, AND THE CONSEQUENT IS NEGATED $(\neg C_{H1})$).
3. REPEAT THESE TWO STEPS FOR (ANY ADDITIONAL HIDDEN LAYERS AND) THE OUTPUT LAYER.

For this net, table 3.1 shows the complete set of rules resulting from all possible combinations of inputs[5].

Although Fu discusses *translating* the neural net into rules, he is really *compiling* a rule base from net knowledge. Alternatively, this algorithm suggests a means of perfoming a translation for every input vector. As shown in Table 3.1, meaningful information can be

[5]The table considers only those rules where all inputs are present (as either an asserted or negated antecedant). Inputs that are not strong enough to affect the resulting output class become "don't cares", and can be ignored. Fu considers such rules to be more *general*; fewer input features are restricted to being either a **1** or a **0**.

Inputs	Neuron Rule	Activation	Class	Conclusion
$I_1 = 0$	$\neg I_1 \wedge \neg I_1 \rightarrow H_1$	$0 < 0.3$	$H_1 \in class0$	$\neg H_1$
$I_2 = 0$	$\neg I_1 \wedge \neg I_2 \rightarrow H_2$	$0 < 0.7$	$H_2 \in class0$	$\neg H_2$
	$\neg I_1 \wedge \neg I_2 \rightarrow H_3$	$0 > -0.2$	$H_3 \in class1$	H_3
	$H_3 \wedge \neg H_1 \wedge \neg H_2 \rightarrow O_1$	$0.3 > 0.1$	$O_1 \in class1$	O_1
	$H_3 \wedge \neg H_1 \wedge \neg H_2 \rightarrow O_2$	$0.3 > -0.1$	$O_2 \in class1$	O_2
	$\therefore \neg I_1 \wedge \neg I_2 \rightarrow O_1 \wedge O_2$	Input: binary 00 Output: binary 11, class 3		
$I_1 = 0$	$I_2 \wedge \neg I_1 \rightarrow H_1$	$0.4 > 0.3$	$H_1 \in class1$	H_1
$I_2 = 1$	$I_2 \wedge \neg I_1 \rightarrow H_2$	$0.3 < 0.7$	$H_2 \in class0$	$\neg H_2$
	$I_2 \wedge \neg I_1 \rightarrow H_3$	$-0.1 > -0.2$	$H_3 \in class1$	H_3
	$H_1 \wedge H_3 \wedge \neg H_2 \rightarrow O_1$	$0.4 + (-0.1) > 0.1$	$O_1 \in class1$	O_1
	$H_1 \wedge H_3 \wedge \neg H_2 \rightarrow O_2$	$(-0.5) + 0.3 < -0.1$	$O_2 \in class0$	$\neg O_2$
	$\therefore \neg I_1 \wedge I_2 \rightarrow O_1 \wedge \neg O_2$	Input: binary 01 Output: binary 10, class 2		
$I_1 = 1$	$I_1 \wedge \neg I_2 \rightarrow H_1$	$-0.2 < 0.3$	$H_1 \in class0$	$\neg H_1$
$I_2 = 0$	$I_1 \wedge \neg I_2 \rightarrow H_2$	$0.5 < 0.7$	$H_2 \in class0$	$\neg H_2$
	$I_1 \wedge \neg I_2 \rightarrow H_3$	$0.3 > -0.2$	$H_3 \in class1$	H_3
	$H_3 \wedge \neg H_1 \wedge \neg H_2 \rightarrow O_1$	$0.3 > 0.1$	$O_1 \in class1$	O_1
	$H_3 \wedge \neg H_1 \wedge \neg H_2 \rightarrow O_2$	$0.3 > -0.1$	$O_2 \in class1$	O_2
	$\therefore I_1 \wedge \neg I_2 \rightarrow O_1 \wedge O_2$	Input: binary 10 Output: binary 11, class 3		
$I_1 = 1$	$I_1 \wedge I_2 \rightarrow H_1$	$(-0.2) + 0.4 < 0.3$	$H_1 \in class0$	$\neg H_1$
$I_2 = 1$	$I_1 \wedge I_2 \rightarrow H_2$	$0.5 + 0.3 > 0.7$	$H_2 \in class1$	H_2
	$I_1 \wedge I_2 \rightarrow H_3$	$0.3 + (-0.1) > -0.2$	$H_3 \in class1$	H_3
	$H_2 \wedge H_3 \wedge \neg H_1 \rightarrow O_1$	$0.2 + 0.3 > 0.1$	$O_1 \in class1$	O_1
	$H_2 \wedge H_3 \wedge \neg H_1 \rightarrow O_2$	$0.4 + 0.3 > -0.1$	$O_2 \in class1$	O_2
	$\therefore I_1 \wedge I_2 \rightarrow O_1 \wedge O_2$	Input: binary 11 Output: binary 11, class 3		

Table 3.1 **Fu Rules.** For the neural net in Figure 3.1, all possible combinations of *binary* inputs are translated into a complete set of *specific* rules.

generated in this manner. While this seems to be a plausible method, several limitations exist, to this modification as well as Fu's method in general.

3.3 Problems/Limitations

3.3.1 Special Architecture. The completeness of the algorithm, Fu suggests, can be considerably improved by using a special procedure to train the neural network[5]. In [4], he describes a special training procedure to accomplish this, which involves clustering hidden units and nullifying small weights. In other words, a special net architecture provides the infrastructure for his rule generation. As noted in Section 2.2, special net architectures limit the generality of the expert network and the rule-generation mechansim.

3.3.2 Ad Hoc Approach. Injecting rules into a neural network is fairly straightforward; interpreting connections and weights in a meaningful manner poses a much greater challenge. Fu's method of extracting rules becomes an exhaustive search (either explicit or implicit with heuristics) of the rule space.

In[5], Fu expresses a series of pruning heuristics to find a consistent set of rules more quickly. However, this method remains inherently ad hoc. The number of rules generated for each neuron is limited by an arbitrary k-wide beam search; a full set of rules (every possible combination of positive and negated attributes) would result in an exorbitant number of rules at each neuron. Folding rules to contain only input conditions in the premise and output actions in the consequent would otherwise become an impossible task.

Fu freely admits that in some domains all available attributes must be involved in any decision and general rules simply do not exist[5].

3.3.3 Discrete Inputs and Outputs. Another limitation is the implicit assumption that all inputs and outputs are binary. As already noted, a steep enough sigmoid function or a hard limiter can provide this binary output response. However, how general is this? A more general output classification approach is a *winner-take-all* strategy, with one output node per output class. In this case, the outputs are *not* discrete. Rather, the output whose value is largest determines the output class. There is no midpoint, nor boundary,

R1	**IF** petal-length ≤ 2.7	**THEN** iris is setosa
R2	**IF** $2.7 <$ petal-length $\leq 5.0 \wedge 0.7 <$ petal-width ≤ 1.6	**THEN** iris is verisicolor
R3	**IF** petal-length > 5.0	**THEN** iris is viginica
R4	**IF** petal-width > 1.6	**THEN** iris is viginica
R5	**IF** septal-length $> 3.1 \wedge 2.7 <$ petal-length ≤ 5.0	**THEN** iris is versicolor

Table 3.2 **Fu's Iris Rule Set.** For the iris data set, Fu extracted only these rules[5].

nor cut-off between "firing" and "not" firing. Quite the contrary. An output of, say, 0.3 can identify the output class if this value is larger than all other outputs; similarly, a large value, say 0.8, may not cause a an output "1" (i.e. a larger output can and will determine the class).

Even if the binary encoded output is acceptable, binary inputs are harder to justify. Even with a steep sigmoid/hard limiter, this only applies to inputs that come from hidden node outputs. Net inputs will typically be real-valued, and they may or may not be between **0** and **1**. With some combination of normalization, quantization, and binarization, it is possible to map the real-valued features to a sequence of binary inputs. Theoretically, the net can be trained normally[6] and rules can be generated. However, precision is lost in quantization, based on the number of bins used.

3.4 Meta Knowledge

Finally, Fu draws conclusions that are not arrived at naturally from his rule-generation process in [5]. Consider his iris results, summarized in Table 3.2. In terms of input binarization, the inequalities suggest a clustering of the inputs into a number of "bins", such as

$$\left|\begin{array}{c|c|c} \text{petal-length} \leq 2.7 & 0.7 < \text{petal-width} \leq 1.6 & \text{septal-width} > 3.1 \\ 2.7 < \text{petal-length} \leq 5.0 & \text{petal-width} > 1.1 & \\ \text{petal-length} > 5.0 & & \end{array}\right| \tag{3.7}$$

For example, rule **R1** indicates the existence of a single bin containing all test vectors with petal-length ≤ 2.7. While this may, in fact, occur, the input presented to the net is

[6] With the added binarization preprocessing step for each input vector.

the binary representation of the bin. The number of bins (clusters) dictates the number of net inputs:

$$\#\text{inputs} = \lceil \log_2 \#\text{bins} \rceil; \tag{3.8}$$

Conversely, the number of net inputs available can determine the maximum number of clusters. Figure 3.2 shows a simulated clustering of the iris training data among two input features. With three bins, $\lceil \log_2 3 \rceil = 2$ inputs are required. An input vector falling into the first bin would be presented to the net as, say, binary **00**. The resultant rule, such as

$$\textbf{IF} \neg A_{I1} \wedge \neg A_{I2} \textbf{THEN } C_{O1}. \tag{3.9}$$

can be mapped back into "real" space by substituting the mean vector feature values:

$$\textbf{IF} \text{ petal-length} = \alpha_{00} \wedge \text{petal-width} = \beta_{00} \textbf{ THEN } C_{O1}. \tag{3.10}$$

This rule is fundamentally concrete. To draw Fu's abstract inequality,

IF petal-length ≤ 2.7 **THEN** iris is setosa,

one must conclude that meta-level knowledge interpretation is taking place of which Fu fails to mention. In the input plot in Figure 3.2, it is not difficult to see *graphically* or *at a higher level* that a boundary for input class "00" exists at about 2.7. Likewise, from a series of rules such as

R1: **IF** petal-length = 2.7 **THEN** iris is setosa
R2: **IF** petal-length = 2.5 **THEN** iris is setosa
R3: **IF** petal-length = 1.8 **THEN** iris is setosa

one can draw the higher level concept

R1: **IF** petal-length ≤ 2.7 **THEN** iris is setosa.

3.5 Bottom Line

Fu does not provide enough implementation information to verify his results. Although assumptions can be made and success can be achieved under certain conditions,

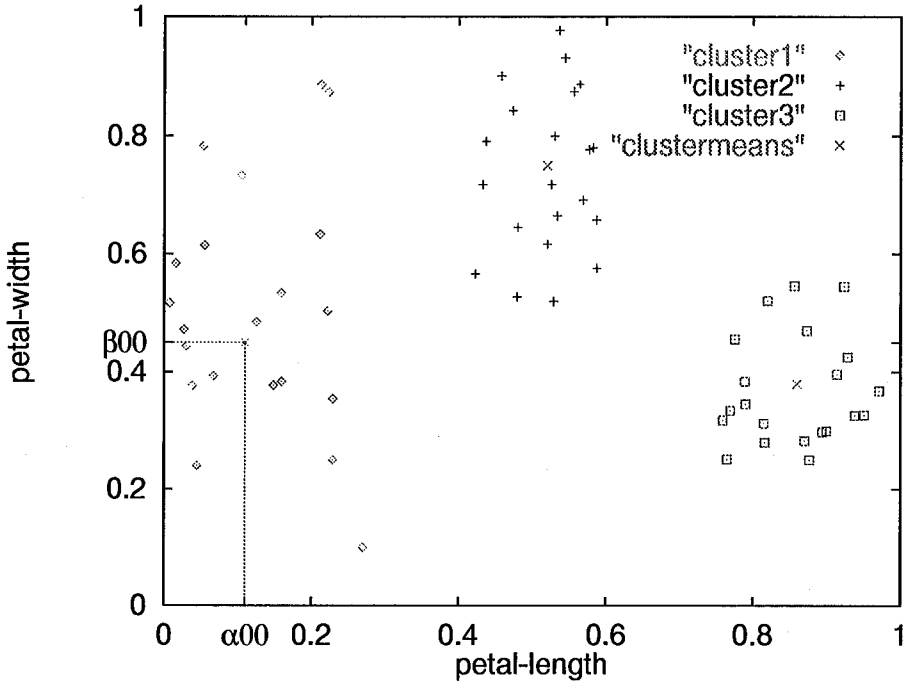


Figure 3.2 **Two-Feature Iris Clustering.** The three clusters represent three input bins; for any input vector, the corresponding bin's binary code is presented to the net. But the binary value can only be translated back into the mean vector feature values. Thus, precision is lost.

the steep sigmoid model is not terribly general. Isn't there a way to expand upon Fu's efforts—recognizing the importance of the mathematical nature of the net knowledge—to form a more general net-interpretation model?

IV. Knowledge Mathematics

Fu begins with the neuron activation function

$$\sum_{i=1}^{n} w_i x_i + \theta = 0, \tag{4.1}$$

imposing the restriction that all inputs, x_i, must be binary. What happens when the inputs vary through some range, say (0,1)[1]? In particular, can this lead to a viable rule-extraction technique, with which to augment a general neural net with an explanation capability?

4.1 Concept Overview

Despite its limitations, Fu's method has some definite strong points. In particular, it attempts to utilize the actual, subsymbolic knowledge held in the inter-neural connections. However, it only explores one aspect of the net's knowledge—the neuron activation function. Furthermore, it contorts these individual bits of knowledge, coercing the activation function into producing the kind of information Fu wishes to see.

In reality, the net knowledge is much more complicated than Fu accounts for. In general, neuron inputs and outputs are *not* binary. Therefore, in interpreting the net knowledge, it is desirable to account for the real-valued nature of these inputs and outputs. Also, the neuron activation is typically squashed by some neuron output function—typically a sigmoid or hyperbolic tangent—that does *not* act as a binary encoder[2].

Perhaps it is possible to extend Fu. More fully exploiting the mathematical associations within the net might provide a more exact symbolic knowledge representation. Such a method would be inherently less ad hoc, less arbitrarily approximate.

---

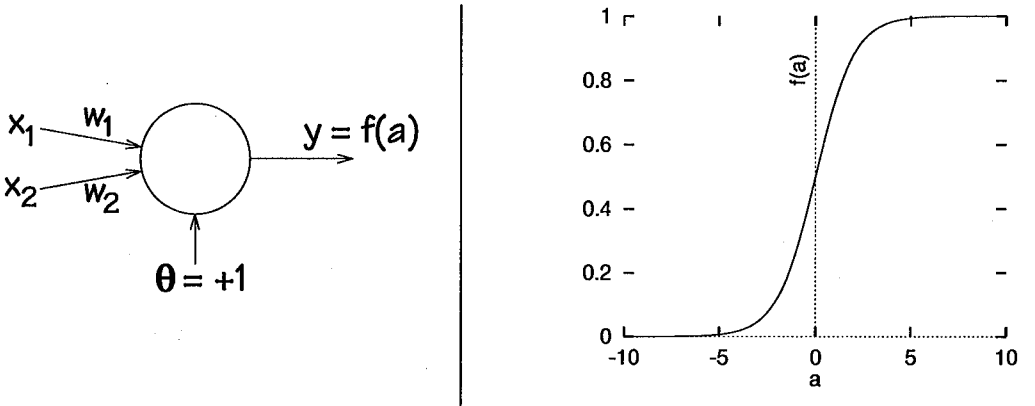
[1]This range bounds the outputs of sigmoidal neurons and, thus, any inputs into which these nodes feed (e.g. for all but the first hidden layer nodes). Net inputs, on the other hand, range through $(I_{\min}, I_{\max})$.

[2]Actually, the neuron output may be "squashed" with a linear function. In this case, the output of the neuron is simply the neuron activation. In fact, it has been shown that the most general MLP configuration employs sigmoids on the hidden layer and linear neurons on the output layer[]dr. rogers conversation–who for real?. However, linear neurons do not work well without sigmoidal or tanh nodes. In particular, linear neurons on the first hidden layer are superfluous (sounds like another great cite–but who?).

4.2 Intuition

Consider the simple perceptron[3] in Figure 4.1a. Assuming a standard sigmoid, with $\lambda = 1$, the neuron output equation is

$$y = \frac{1}{1 + e^{-}(w_1 x_1 + w_2 x_2 + \theta)}. \tag{4.2}$$



(a) Simple 2-Input Perceptron

(b) The Sigmoid Function

Figure 4.1 **Single-Node Perceptron.** The output of the net in (a) is a function of the neuron activation: $y = f(w_1 x_1 + w_2 x_2 + \theta)$. A common output function is the sigmoid, whose behavior is shown in (b)

In this case, a binary encoded output is not unjustified. The utility of a neural net is contingent upon it realizing at least two classes. With only one output neuron (indeed, only one neuron, period!), the perceptron output *must* be quasi-binary. That is, the sigmoid midpoint (0.5) represents the boundary between classes (see Figure 4.1b). Since the output function passes through 0.5 when the activation goes through zero, the activation function provides all the information needed to interpret the net behavior.

[3]A perceptron is the simplest net, consisting of a single neuron or a single layer of neurons. This basic architecture can correctly classify any *linearly separable* data[17]; more complex functions require more intricate architectures, such as the *multi-layer* perceptron used throughout this thesis.

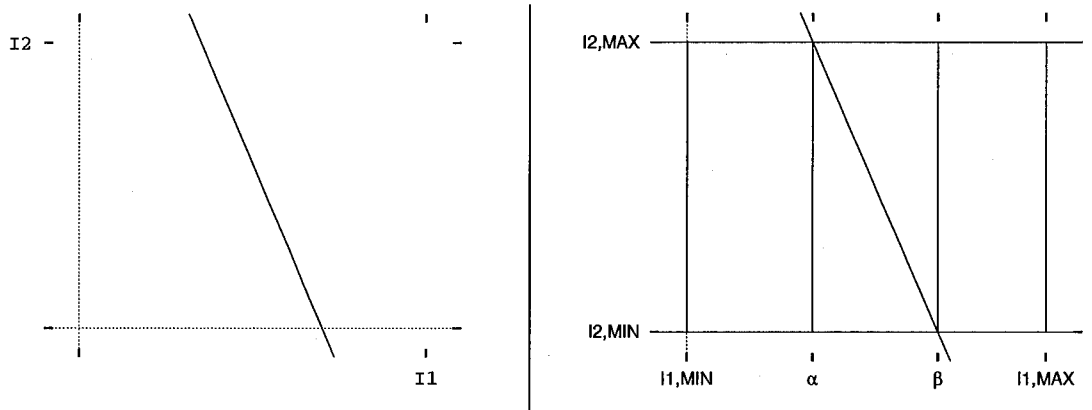
The activation function is

$$0.5x_1 + 0.5x_2 - 1 = 0. \tag{4.3}$$

Note that this equation has the form

$$Ax + By + C = 0. \tag{4.4}$$

That is, the activation function is linear in terms of the inputs. Plotting this function, one can readily observe that it divides the input space into two sections, as in Figure 4.2a.



(a) Activation Line **(b)** Rule Regions

Figure 4.2 **Perceptron Activation Plot.** The neuron activation inscribes a line through the input space in (a), effecting the three rule regions shown in (b).

In this example, the *rule space* is partitioned into three distinct *rule regions* (see Figure 4.2b). When $i_1 < \alpha$, the output belongs to *Class 0* regardless of the value of i_2. Similarly, if $i_1 > \beta$, the neuron output ($O \in$ *Class1*) depends only on the first input. However, when $\alpha < i_1 < \beta$, the output could belong to *Class 0* or *Class 1*, depending upon which side of the *activation function* line the input vector lies. Thus, an input in this region depends on a *relationship* between i_1 and i_2, determined by the activation function. The defining rules are summarized in table 4.1.

Region	Dependency	Rule
$i_1 < \alpha$	Output depends only on i_1	Simple:**IF** $i_1 < \alpha$ **THEN** $\neg O$
$i_1 > \beta$	Output depends only on i_2	Simple:**IF** $i_1 > \beta$ **THEN** O
$\alpha < i_1 < \beta$	Output depends on i_1 and i_2	Complex: **IF** $i_1 > \alpha \wedge i_1 < \beta \wedge i_1 < -\frac{w_2}{w_1} i_2 - \frac{\theta}{w_1}$ **THEN** $\neg O$
		Complex: **IF** $i_1 > \alpha \wedge i_1 < \beta \wedge i_1 > -\frac{w_2}{w_1} i_2 - \frac{\theta}{w_1}$ **THEN** O

Table 4.1 **Rule Dependencies.**

The activation line disects the input space *bounding box.* The bounding box is the input region bounded between $i_{1,\min}$ and $i_{1,\max}$ in the i_1 direction, and $i_{2,\min}$ and $i_{2,\max}$ along the i_2 feature axis. For all neurons but those in the first hidden layer, the bounding box inscribes a square (for two dimensions) from 0 to 1[4]. Note that the activation line cuts this box in one of four ways (illustrated in Figure 4.3):

1. If the activation line traces a side of the box, or lies entirely outside the box (touches the box **0**, 1 or many times), it does not *cross* any side. In this case, the output is constant (always **1**, or always **0**).
2. If the line crosses exactly two sides (any two), the input space is partitioned into three rule regions. Two of the regions are bounded by only one input. The third realizes a relationship between the two inputs.
3. If the line cuts one side and one corner, the bounding box is cut into two regions: one is bounded by one input; the other is a relationship between the two inputs.
4. If the line passes through two opposite corners (which can be viewed as crossing all four sides), the entire bounding box represents a single region whose rule is an input relationship.

It is possible to generate Fu-like rules from the net in the following manner. Suppose Figure 4.2 represents a solution to a two-input subset of the iris problem. Letting i_1 be *petal length* and i_2 be *petal width*, the effective input feature ranges and the activation line

[4]This assumes a sigmoid output function. A tanh node is "bounded" between -1 and 1.

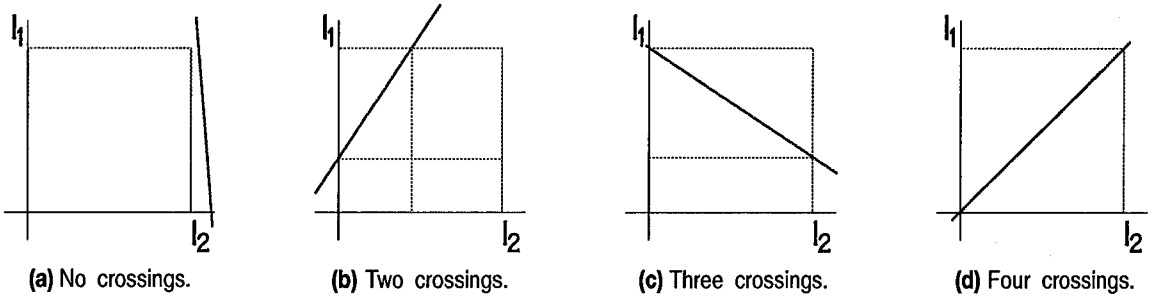


Figure 4.3 **Rule Space Partitioning.** The activation line separates the input space into (a) 1, (b) 3, (c) 2, and (d) 1 rule regions. Some regions generate simple rules; others are complex input relationships.

behavior, illustrated in Figure 4.4, might be[5]:

$$\begin{array}{rcl||rcl} i_{1,\min} & = & 1.2 & i_{1,\max} & = & 6.2 \\ i_{2,\min} & = & 0.7 & i_{2,\max} & = & 1.6 \\ \alpha & = & 2.7 & \beta & = & 5.0 \end{array} \tag{4.5}$$

Two of Fu's iris rules (see Section 3.4) can be read directly from Figure 4.4:

$$\begin{array}{l} \textbf{IF} \text{ petal-length } < \ 2.7 \ \textbf{THEN} \text{ iris is setosa} \\ \textbf{IF} \text{ petal-length } > \ 5.0 \ \textbf{THEN} \text{ iris is viginica} \end{array} \tag{4.6}$$

An additional rule, describing the complex region $2.7 < i_1 < 5.0$, turns out to be a somewhat more complicated relation between i_1 (petal length) and i_2 (petal width):

$$\begin{array}{ll} \textbf{IF} & \text{petal-length} > 2.7 \\ & \wedge \text{petal-length} < 5.0 \\ & \wedge \text{petal-width} > 0.7 \\ & \wedge \text{petal-width} < 1.6 \\ & \wedge \textbf{petal-length} > \textbf{(0.4)petal-width}-\textbf{0.36} \\ \textbf{THEN} & \text{iris is verisicolor} \end{array} \tag{4.7}$$

Therefore, it would appear that the neuron activation, without arbitrary restrictions on the neuron inputs, might be sufficient to extract semantically viable rules. Moreover,

[5]The effective input ranges can be ascertained from the training data. Training data that is representative of the problem domain will result in accurate input ranges. This illustrates one aspect of the importance of an appropriately general training set.

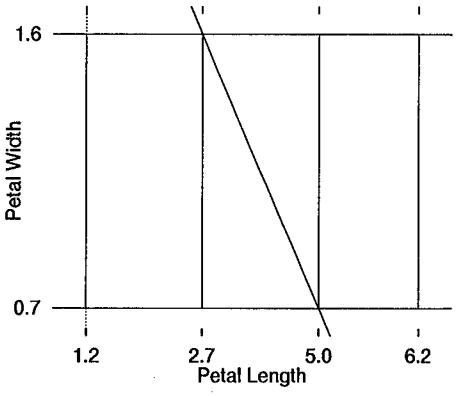
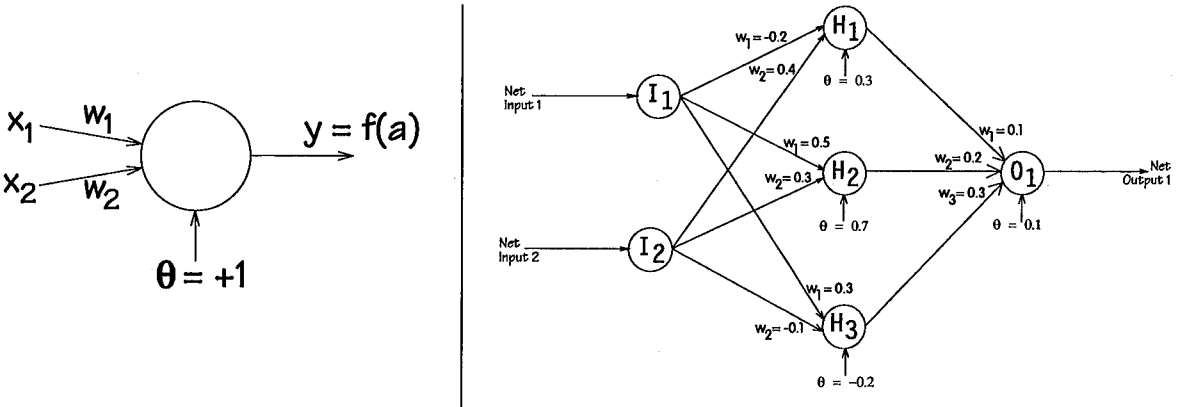


Figure 4.4 **Two-Input Iris Activation.** A two-input subset of the iris problem might produce an activation such as this, relating inputs, *petal length* and *petal width* to output classes (e.g *setosa, versicolor,* or *viginica*).



(a) Simple 2-Input Perceptron

(b) Multi-Layer Perceptron

Figure 4.5 **Perceptron Architectures.** A simple perceptron in (a) containing only one neuron vs. a *multi-layer* perceptron with one hidden layer in (b).

because these rules utilize more of the embedded neural net characteristics, they may better represent the subsymbolic knowledge than those discovered via Fu's method.

Does this concept extend to a general multi-layer perceptron?

Consider the multilayer perceptron (MLP) with one hidden layer in Figure 4.5b. Like the simple perceptron, all neuron activations are linear with respect to the neuron inputs[6].

For the hidden nodes, the activation is a function of the network inputs i_1 and i_2. Considering only the neuron activations, each hidden node partitions the input space into one or two sections (up to three rule regions), according to the guidelines in Section 4.2. With one such activation line per hidden node, the total number of input-space lines equals the number of hidden nodes N_h. The hidden node activations that cross the bounding box help partition the input space into intuitive rule regions (see Figure 4.6a).

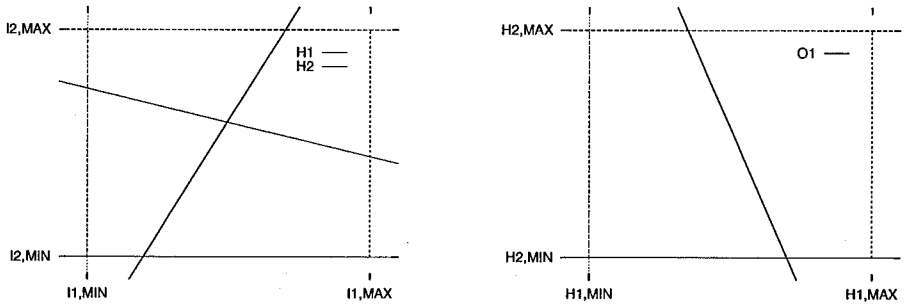
Similarly, the output layer node activations (only 1, in this case) realize linear partitions of the "hidden" space (Figure 4.6b). Is it possible to conflate the two and identify the consequential input regions inscribed by the hidden node activations, as implied by (Figure 4.6c)?

Figure 4.7a shows an MLP to ideally classify linearly separable data[7]. In this case, all data vector values lie within the range (0,1) with a class separation as shown in Figure 4.8a.

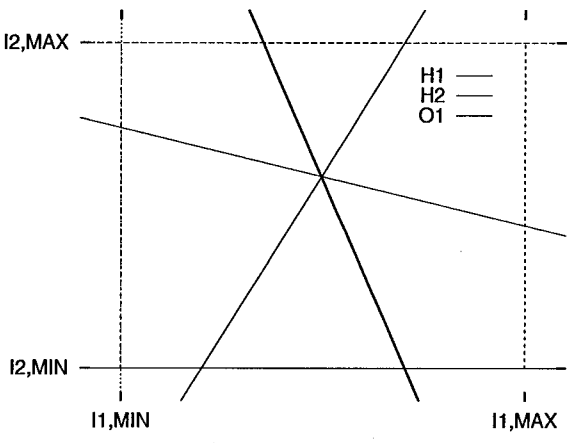
This "perfect" net results in an absolute bounding of the implicit rule regions. Graphically, one hidden node separates the "top" data from the "bottom", and the other separates "left" data from "right". Together they divide the input space into four potential rule regions (Figure 4.7b). The output node activation line cuts diagonally through the hidden space bounding box (Figure 4.7c). Figure 4.8b shows the input data with the hidden node activations and the superimposed diagonal output line. Note that the combination of hidden and output activations correctly and efficiently isolates the two data classes.

[6] Actually, planar (or even hyperplanar) in terms of an general n-dimensional neuron input space.

[7] The simple perceptron in Figure 4.5a can correctly classify this data much more efficiently; this architecture is presented for illustrative purposes only.

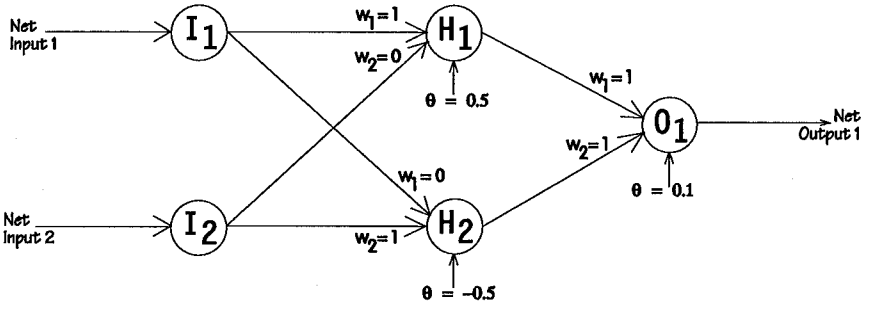


(a) Hidden Node Activations. **(b)** Output Node Activations.

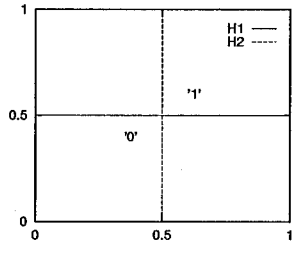


(c) Superimposed Output Activation.

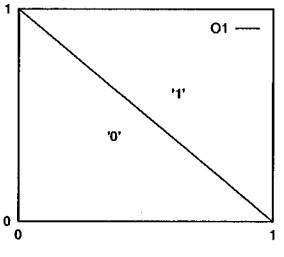
Figure 4.6 **MLP Activations by Layer.** Each node of the MLP in Figure 4.5 has one activation line. Thus, the hidden layer inscribes two lines through the input space, (a), and the output node activation traces one line through the "hidden" space, (b). In (c), the output activation is superimposed over the input space, suggesting a means of distinguishing pertinent rule regions.



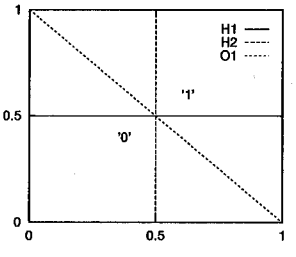
(a) "Perfect" Linear Separator.



(b) Hidden Layer.

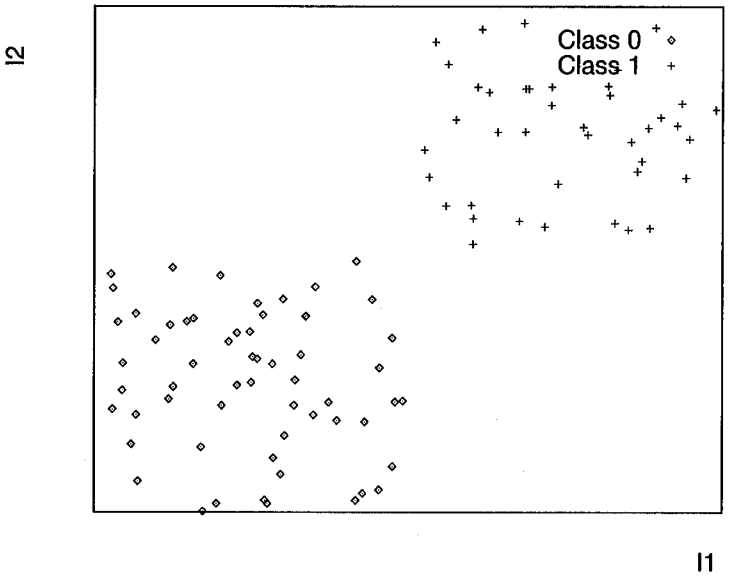


(c) Output Layer.

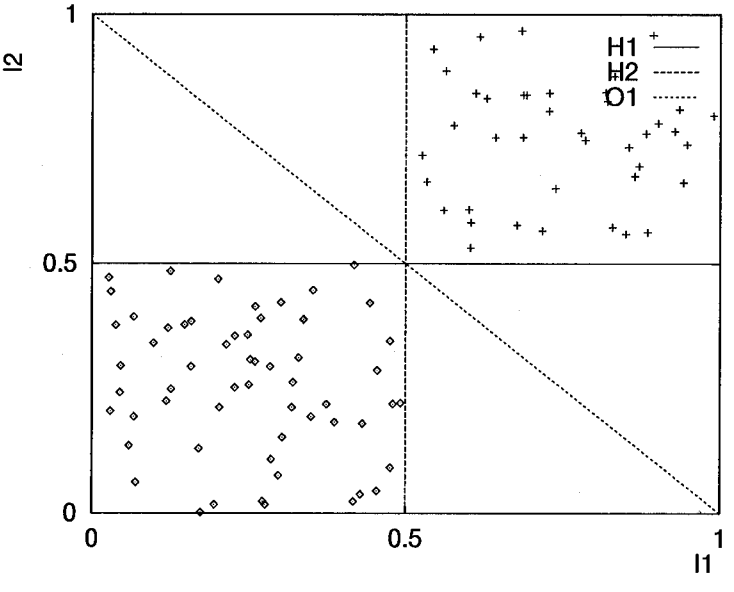


(d) Superimposition.

Figure 4.7 **Linearly Separable by Hand.** An ideal MLP can be constructed by hand to partition the linearly separable data in Figure 4.8 such that the hidden and output activations occur as in (b) and (c), respectively. In this case, the superposition of the output activation over the hidden lines (in (d)) clearly discriminates *class 0* from *class 1*.



(a) Sample Data



(b) Ideal Linearly Separable Data Bounding

Figure 4.8 **Linearly Separable Data.** Since the two clusters of data in (a) can clearly be separated with a straight line, a simple perceptron is sufficient to classify this domain. In (b), the hidden and output activations of the MLP in Figure 3.3.3 tightly bound and correctly classify this data.

Figure 4.9 shows the activations of an actual 2-2-1 MLP trained against this data set. Although the regions bounded by the activations are not as clean as the handmade case, the output classification is nicely bounded none-the-less[8].

This appears too good to be true! Something must be wrong, or semantically interpreting the subsymbolic net would be straightforward.

4.3 Problems

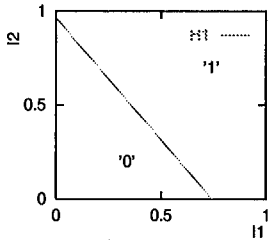
The problem, of course, is that the neuron activation is squashed by a nonlinearity function, typically a sigmoid or tanh. The sigmoidal neuron output is

$$y = \frac{1}{1 - e^{\sum_{i=1}^{n} w_i x_i + \theta}}. \tag{4.8}$$

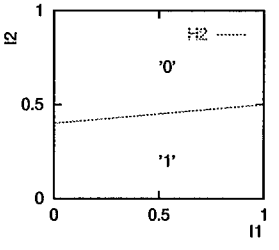
Recall the simple perceptron activation of Section 3.3.3, plotted again in Figure 4.10a. Figure 4.10b shows the neuron output, y, plotted against its two inputs, i_1 and i_2. The exponential term "shifts" the activation line, creating a region of uncertainty about the midpoint. That is, because of the relative steepness of this part of the graph, it is hard to distinguish precisely where one class ends and the other begins. Input vectors near this boundary are subject to misclassification.

The magnitudes of the weights, w_1 and w_2, and the neuron bias, θ, determine the location and slope of the activation line. Note that these correspond to the coefficients of the general line equation ($A = w_1, B = w_2$, and $C = \theta$). Furthermore, the side of the activation line corresponding to each output class depends on the *sign* of the weights. Consider the following two sets of constraints and the resulting activation equation:

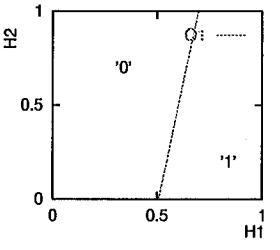
[8] Actually, Figure 4.9 shows that hidden node H_1 is doing most of the work, not surprising since a single neuron can discriminate linearly separable data.



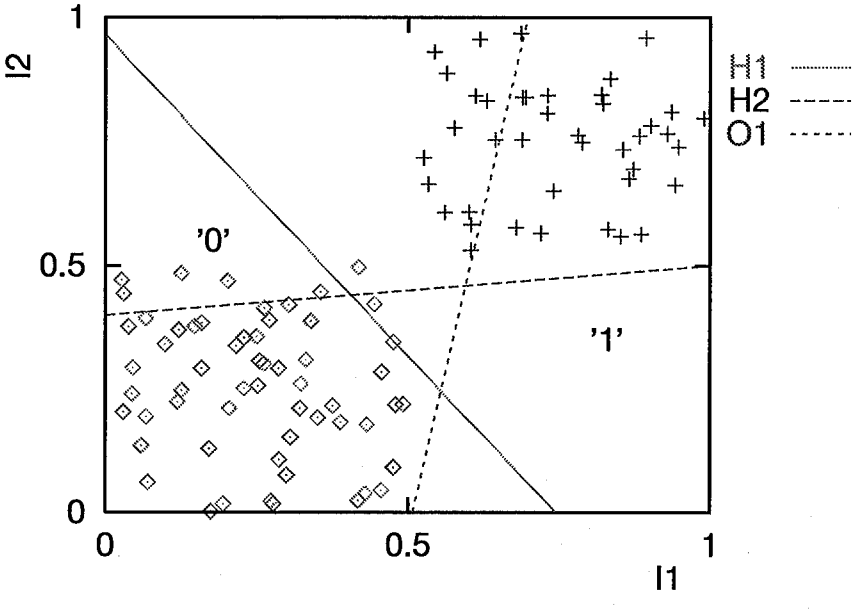
(a) $I_2 < -1.30I_1 + 0.96 \rightarrow \neg O$
$I_2 > -1.30I_1 + 0.96 \rightarrow O$



(b) $I_2 > 0.10I_1 + 0.40 \rightarrow \neg O$
$I_2 < 0.10I_1 + 0.40 \rightarrow O$

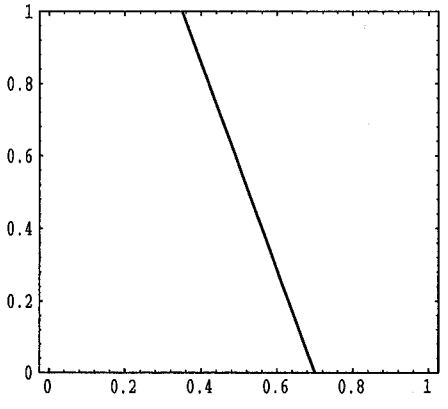


(c) $H_2 < 5.28H_1 - 2.68 \rightarrow \neg O$
$H_2 > 5.28H_1 - 2.68 \rightarrow O$

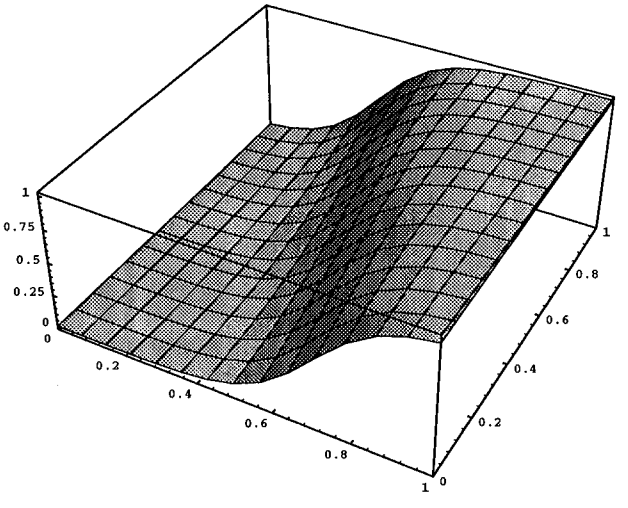


(d) Logical *class 0* and *class 1* regions.

Figure 4.9 **Linearly Separable by Training.** Figure 4.7 suggested superimposing the output activation over the input space could isolate the output class clusters. However, a real application produces the counterintuitive regions as demonstrated here.



(a) $w_1 I_1 + w_2 I_2 + \theta = 0$



(b) $y = \frac{1}{1+e^{-w_1 I_1 - w_2 I_2 - \theta}}$

Figure 4.10 **Neuron Output Behavior.** A neuron activation function inscribes a line through the feature space in (a). This line corresponds to the midpoint of the squashed output in (b).

$$
\begin{array}{rcl|rcl}
w_1 & = & 1 & w_1 & = & -1 \\
w_2 & = & 1 & w_2 & = & -1 \\
\theta & = & -1 & \theta & = & 1 \\
\multicolumn{3}{c|}{} & \multicolumn{3}{c}{} \\
i_1 + i_2 - 1 & = & 0 & -i_1 - i_2 + 1 & = & 0 \\
\multicolumn{6}{c}{} \\
\multicolumn{6}{c}{i_2 = -i_1 + 1}
\end{array}
\tag{4.9}
$$

Although both activations trace the same line through the $0-1$ bounding box (Figure 4.11a), the neuron output plots (Figures 4.11b,c) clearly show that flipping the signs of the weights and biases is sufficient to flip the side of the activation line corresponding to *class 0* and *class 1*.

Thus, the rule regions are not as clearly defined as Section 4.2 suggested, and conflation of multiple layers is not graphically obvious.

Perhaps mathematics may provide answers that the graphs alone cannot.

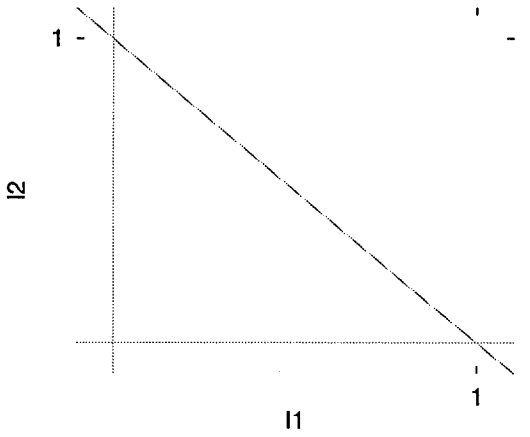
4.4 Mathematical Derivation

It behooves one, then, to attack the interpretation problem using the precise math of the net, avoiding the need to predict or approximate rule regions based on offseting activation functions. A general net-interpretation technique must account for the neuron squashing functions as well. Still, as Fu might contend, binary outputs provides a good place to start.

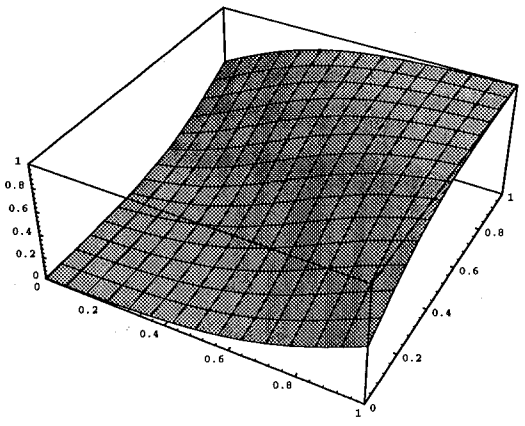
4.4.1 Binary Output. Consider an arbitrary sigmoid-sigmoid MLP with N output neurons. Assuming the outputs are binary[9], the N outputs can realize up to 2^N output classes. Like the simple perceptron, the output neuron sigmoid can be ignored; the 0.5 midpoint represents the cut-off between (borrowing from Fu) the neuron "firing" and "not firing". In other words:

$$
\begin{array}{rcl}
y_{O_i} > 0.5 & \rightarrow & \sum_{i=1}^{n} w_i x_i + \theta > 0 \quad \rightarrow \quad O_i \triangleq 1 \\
y_{O_i} < 0.5 & \rightarrow & \sum_{i=1}^{n} w_i x_i + \theta < 0 \quad \rightarrow \quad O_i \triangleq 0
\end{array}
\tag{4.10}
$$

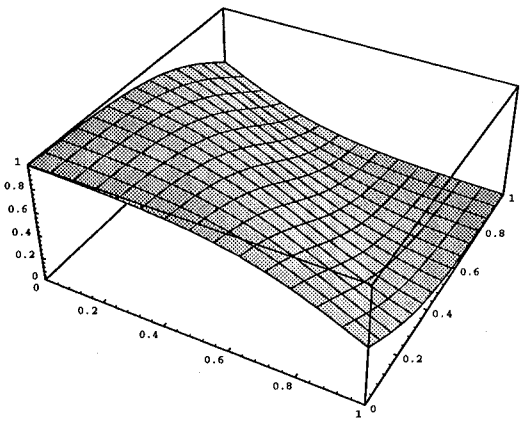
[9]As explained previously, forced binary outputs yield a less general net. However, for the sake of illustration, this is a good place to begin. Section 4.4.4 will look at the more general, non-binary outputs.



(a) Activation line corresponding to: $\begin{aligned} x + y - 1 &= 0 \\ -x - y + 1 &= 0 \end{aligned}$



(b) Neuron output: $y = \frac{1}{1+e^{-(x+y-1)}}$



(c) Neuron output: $y = \frac{1}{1+e^{-(-x-y+1)}}$

Figure 4.11 **Squashed Output Function.** Despite inscribing the same activation line, the activation equations in (a) realize opposite output classifications, (b) and (c).

This simplification results not from any arbitrary approximation, but from knowledge of the neural net functionality. The advantage is that this corresponds to the activation function passing through zero, and the activation is linear with respect to the hidden nodes. Thus, the need to consider the sigmoid exponential term in the output layer is bypassed.

Again, the line inscribed by the activation equation partitions the "hidden" space bounding box into as many as three rule regions (see Section 4.2 to recap). Unless the line fails to cut the bounding box[10], the rule space will be partitioned into one "general" rule region, and zero, one, or two "shortcut" regions.

This concept may seem counterintuitive. In symbolic expert system terminology, "general" means less restrictive. Bounding input ranges is clearly more restrictive than letting them range freely; hence, a region bounded by a single input (i.e. only one input range is bounded) provides a more general rule than one in which two (or more) input ranges are tied. However, while the *rule* associated with a short-cut region may be more general, the *region* typically covers a small section of the input space. In fact, the area covered by short-cut rules generally decreases as the dimensionality increases.

Suppose Figure 4.11a describes the activation of the simple two-input perceptron (Figure 4.5a). If the perceptron contains three inputs instead of two, then the *bounding box* becomes a *bounding cube*, through which an *activation plane* passes (see Figure 4.12a). Figure 4.11a becomes one side of the cube say, when $i_3 = 0$). This activation plane can either:

pass "straight through", cutting the opposite side exactly the same, or
"slant" to one side or the other, cutting the opposite side parallel to the first.

Figures 4.12b and c demonstrate this slanting effect, where the area governed by simple regions (bounded by just one input shrink with respect to an extra dimension. Figure 4.12b shows the activation equation of the two-input perceptron, now representing the side of the bounding *cube* corresponding to $i_3 = 0$. On this side of the cube, the first simple region is bounded by $i_1 < \alpha$. But Figure 4.12c shows the opposite side of the cube ($i_3 = 1$). Notice that the area bounded by the first region shrinks to $i_1 < \gamma$. Between γ

[10]Trivially, if the activation line does not cross the bounding box, the rule space contains one simple rule region—the output is always *1* or always *0*, regardless of either input value.

and α, the input vector lies within the cube in an area in which the value of i_3 will affect the resulting output class.

As "special case" regions, the shortcut regions lend themselves well to straightforward rule generation; hence, they provide a good place to attempt symbolic interpretation.

4.4.2 Shortcut Regions. Shortcut rules correspond to rule regions bounded by a single input. For the output layer, the neuron inputs are outputs from hidden nodes. Thus, an output node shortcut region is one bounded by a single hidden node. Consider the output activation with two shortcut rules:

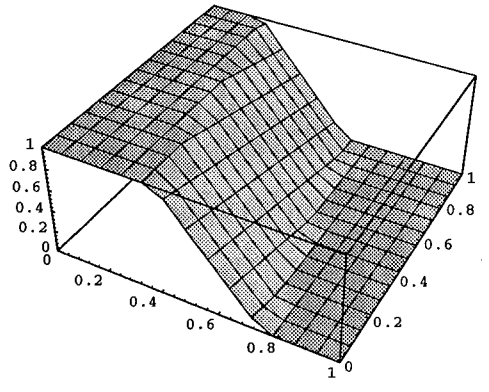
$$H_1 > \alpha \rightarrow O_1 \ (O_1 \in \textit{class 1}) \tag{4.11}$$

and

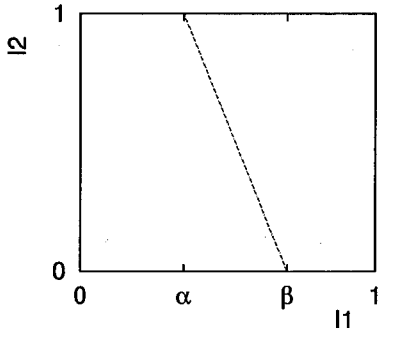
$$H_2 > \beta \rightarrow O_1 \ (O_1 \in \textit{class 1}). \tag{4.12}$$

Writing H_1 (in Equation 4.11) and H_2 (in Equation 4.12) in terms of the inputs, I_1 and I_2, according to the neuron activation functions[11], results in linear input-to-output relationships; semantically viable rules seem to be emerging. However, node O_1's inputs (the outputs from H_1 and H_2) are not binary. To precisely interpret the net knowledge without taking any unnecessary liberties, the hidden node activation function in not sufficient. Accounting for an output sigmoid, Equation 4.11 becomes

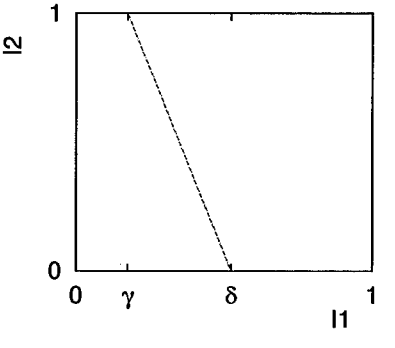
[11]This assumes the binary input restriction imposed by Fu.



(a) Activation plane cutting through a three-input feature space.



(b) I_1-I_2 plane at $I_3 = 0$



(c) I_1-I_2 plane at $I_3 = 1$

Figure 4.12 **Three-Input Activation Plane.** (a) shows an *activation plane* cutting a *bounding cube*. The sides of the cube corresponding to $I_3 = 0$ and $I_3 = 1$ are shown in (b) and (c), respectively.

$$\frac{1}{1+e^{-(w_{H_1 1}I_1+w_{H_1 2}I_2+\theta_{H_1})}} > \alpha \tag{4.13}$$

$$1+e^{-(w_{H_1 1}I_1+w_{H_1 2}I_2+\theta_{H_1})} > \frac{1}{\alpha} \tag{4.14}$$

$$e^{-(w_{H_1 1}I_1+w_{H_1 1}I_2+\theta_{H_1})} > \frac{1}{\alpha}-1 \tag{4.15}$$

$$-(w_{H_1 1}I_1+w_{H_1 2}I_2+\theta_{H_1}) > ln(\frac{1}{\alpha}-1) \tag{4.16}$$

$$w_{H_1 1}I_1+w_{H_1 2}I_2+\theta_{H_1} < -ln(\frac{1}{\alpha}-1) \tag{4.17}$$

$$w_{H_1 1}I_1+w_{H_1 2}I_2 < -ln(\frac{1}{\alpha}-1)-\theta_{H_1} \tag{4.18}$$

Equation 4.18 can be rewritten as

$$w_{H_1 1}I_1+w_{H_1 2}I_2+ln\left(\frac{1}{\alpha}-1\right)+\theta_{H_1}<0. \tag{4.19}$$

Identifying the constant coefficients,

$$\begin{aligned} A &= w_{H_1 1} \\ B &= w_{H_1 2} \\ C &= ln\left(\tfrac{1}{\alpha}-1\right)+\theta_{H_1} \end{aligned} \tag{4.20}$$

it is easy to recognize a *linear* relationship established between net inputs and net outputs. Thus, the output and hidden relationships have been conveniently conflated, despite the sigmoid non-linearity! Equation 4.19 acts as a hidden node activation, similarly disecting the input space into two sections. The inequality in Equation 4.19 dictates which side of this line corresponds to the output class (*class 1* in this case).

Figure 3.3.3 shows the input space partitioned by this pseudo-activation shortcut line. The input space is sectioned into three rule regions; Only two—one shortcut and one general region—apply to this rule, as indicated. Region ① identifies an input space shortcut region in this *shortcut space*, leading to the simplest input-to-output relationship, a short-shortcut rule:

$$I_1 < \gamma \rightarrow O_1. \tag{4.21}$$

The general region (Region ②) demarcates the boundary between $O_1/in\ class\ 0$ and $O_1 \in class\ 1$. The implicit relationship between I_1 and I_2 in this region determines the general shortcut rule

$$[I_1 > \gamma] \wedge \left[I_1 < \frac{1}{w_{11}^1}\left(-ln\left(\frac{1}{\alpha}\right) - \theta_1\right) - w_{21}^1 I_2\right] \rightarrow O_1. \tag{4.22}$$

Thus the hidden space shortcut region, $H_1 < \alpha$, provides simple short-shortcut rules and nice, linear general input relationships relating net inputs to the net output $O_1 \in class\ 1$. A similar derivation will result in clean rule relationships for shortcut region, $H_2 > \beta$. Will this concept extend to the general region of the output activation?

4.4.3 General Region. The complex region indicates an area of the graph relating O_1 to both hidden nodes H_1 and H_2. In this *general region*, the output can belong to either *Class 0* or *Class 1*, depending on which side of the activation line the input falls[12]. Thus, the general region rule has the form

$$[H_1 > \alpha] \wedge [H_2 < \beta] \wedge [w_{O_1 1} H_1 + w_{O_1 2} H_2 + \theta_{O_1} > 0] \rightarrow \neg O_1 \tag{4.23}$$

The simple relations, $H_1 > \alpha$ and $H_2 < \beta$, can be rewritten easily according to the shortcut rules in Section 4.4.2. Technically, these elements of Equation 4.23 are superfluous; recognizing the bounding box crossings (α and β) merely provides a means of identifying simpler—and potentially faster shortcut—rules. The key component in this general rule equation is the activation line, $w_{O_1 1} H_1 + w_{O_1 2} H_2 + \theta_{O_1} > 0$. The bounding box is an artifact of the neuron input ranges. $i_{1,\min}, i_{1,\max}, i_{2,\min}$, and $i_{2,\max}$ will vary for different types of inputs, (i.e. output of a linear or tanh node vs. the sigmoid). Moving the sides of the box (varying $i_{\min}$ and $i_{\max}$ changes the bounding box crossings, but, the neural net has learned the activation line; this line extends indefinitely and will cover any finite crossings. Thus, conflating the hidden space equation into the input space requires focusing only on the activation line. Rewriting the hidden nodes in terms of the inputs yields

[12]Again for clarity: the input vector for this *output* neuron is composed of the output from each hidden node. That is, this neuron's input space corresponds to the net's hidden space.

$$w_{O_1 1} H_1 + w_{O_1 2} H_2 + \theta_{O_1} = 0 \quad (4.24)$$

$$w_{O_1 1} H_1 + w_{O_1 2} H_2 = -\theta_{O_1} \quad (4.25)$$

$$w_{O_1 1} \frac{1}{1 + e^{-(w_{H_1 1} I_1 + w_{H_1 2} I_2 + \theta_{H_1})}} + w_{O_1 2} \frac{1}{1 + e^{-(w_{H_2 1} I_1 + w_{H_2 2} I_2 + \theta_{H_2})}} = -\theta_{O_1} \quad (4.26)$$

$$\frac{1}{w_{O_1 1}} \left(e^{-(w_{H_1 1} I_1 + w_{H_1 2} I_2 + \theta_{H_1})}\right) + \frac{1}{w_{O_1 2}} \left(e^{-(w_{H_2 1} I_1 + w_{H_2 2} I_2 + \theta_{H_2})}\right) = -\frac{1}{w_{O_1 1}} - \frac{1}{w_{O_1 2}} - \frac{1}{\theta_{O_1}} \quad (4.27)$$

The derivation quickly degrades. Because the exponential terms are different, no common logarithm can be applied to Equation 4.27 to get rid of them. Therefore, unlike the shortcut regions, this rule cannot be collapsed mathematically into a nice (e.g. linear) relationship between inputs and output; the relationship between the inputs, I_1 and I_2, and the output, O_1, in this region does not produce easily understandable rules. The resulting rule represents a complicated, convoluted association of I_1 and I_2. What is the semantic value of this?

Despite this limitation, the nice interpretability of the shortcut regions is hard to ignore. The Knowledge Math method has the advantage of not generating an entire, separate rule set; this technique attempts to find applicable net knowledge on a per-input-vector basis. Additionally, this method does not restrict the output classification to be binary encoded.

4.4.4 Winner Takes All. Consider again the MLP in Figure 4.5b. In the winner-takes-all classification scheme, the output neuron with the largest value determines the output class. The *second largest* output value determines the threshold for which the winning neuron will get to classify the vector. For example, suppose the net classifies a particular input vector, $\vec{I_j}$, such that

$$\begin{aligned} O_1 &= 0.7 \\ O_2 &= 0.4 \end{aligned} \quad (4.28)$$

O_1 is greater than O_2; therefore, the output belongs to *Class 1* and not *Class 2*. However, the precise value of O_1 is not paramount; O_1 would "win" for any value $O_1 > O_2$. Thus, the value of O_2 becomes the cutoff threshold for the resulting rule, rather than the sigmoidal midpoint, 0.5. For symbolic interpretation, O_2 can be assumed constant. The

rule then becomes

$$\textbf{IF } O_1 > O_2 \textbf{ THEN output} \in \textit{class 1} \tag{4.29}$$

and the applicable rule regions can be derived as in Section 4.4.1 above, starting with the threshold value.

$$\begin{aligned}
O_1 &= O_2 && (4.30)\\
\frac{1}{1+e^{-(w_{O_1 1}H_1+w_{O_1 2}H_2+\theta_{O_1})}} &= O_2 && (4.31)\\
1+e^{-(w_{O_1 1}H_1+w_{O_1 2}H_2+\theta_{O_1})} &= \frac{1}{O_2} && (4.32)\\
e^{-(w_{O_1 1}H_1+w_{O_1 2}H_2+\theta_{O_1})} &= \frac{1}{O_2}-1 && (4.33)\\
-(w_{O_1 1}H_1+w_{O_1 2}H_2+\theta_{O_1}) &= ln\left(\frac{1}{O_2}-1\right) && (4.34)\\
w_{O_1 1}H_1+w_{O_1 2}H_2+\theta_{O_1} &= -ln\left(\frac{1}{O_2}-1\right) && (4.35)\\
w_{O_1 1}H_1+w_{O_1 2}H_2+ln\left(\frac{1}{O_2}-1\right)+\theta_{O_1} &= 0 && (4.36)\\
& && (4.37)
\end{aligned}$$

Once again, a pseudo-activation line results. This line cuts the hidden space bounding box into two sections and up to three rule regions, just as before. Shortcut regions can be conflated with the input space, furnishing good symbolic rule relations. Thus, for the Knowledge Math rule generation method, the *winner-takes-all* strategy is equivalent to the binary encoded output scheme. It is unfortunate that this technique cannot be easily applied to the complex rule regions.

4.4.5 Observations. The shortcut regions, in fact, *may* be sufficient to approximate the net. When the area bounded by the general rule is small enough[13], the shortcut regions will correctly interpret virtually all input vectors. The complex region can be ignored if the line through the input space is "vertical" enough. This leaves two shortcut regions of interest: Region ① , whose output belongs to *class 0*, and Region ③ , where

[13]The slope of the rule line is either very steep ($m_{\text{rule}} \cong 1$) or very shallow ($m_{\text{rule}} \cong 0$), or the line barely cuts one corner of the bounding box.

Region	Rule Type	Rule
Region ①	short-shortcut:	$I_1 < \gamma \rightarrow O_1$
	general shortcut:	$[I_1 > \gamma] \wedge \left[I_1 < \frac{1}{w_{11}^1}(-ln(\frac{1}{\alpha}) - \theta_1 - w_{21}^1 I_2\right] \rightarrow O_1$
Region ③	short-shortcut:	$I_1 > \delta \rightarrow O_1$
	general shortcut:	$[I_1 < \delta] \wedge \left[I_1 > \frac{1}{w_{11}^1}(-ln(\frac{1}{\beta}) - \theta_1 - w_{21}^1 I_2\right] \rightarrow \neg O_1$

Table 4.2 **Small Complex Region Rule Set.** For this special case, when the area covered by the complex rule region is small, the shortcut rules sufficiently interpret the net. Assuming the activation line in Figure 3.3.3 to be virtually vertical, these "nice" rules completely describe the associated net.

the output is *class 1*. As described above, the shortcut regions provide semantically pleasing input-output rules. Since these shortcut regions describe every possible output class (two, in this case), the shortcut rules provide a viable set of interpretations for *every input vector*. Consider a vector, $\vec{I_j}$, very close to the activation line (i.e. the points lies within Region ②). Because the area covered by Region ② is presumed small, $\vec{I_j}$ must also lie very close to Region ① or Region ③ , and the appropriate shortcut rule will aptly explain the classification of $\vec{I_j}$. Table 4.2 displays the entire rule set associated with the net in this case.

Also, clustering the test data can help determine the likelihood of an arbitrary input vector falling into the general rule region[14]. If the input almost never falls into this complex region, the region can be ignored for rule-generation purposes, no matter how much bounding box it consumes[15].

However, vectors within this general region will not be explained by any rule. As more vectors fall within this region, the resultant symbolic knowledge degenerates into an approximation. There is simply no guarantee that the general region will play an insignificant role.

Ultimately, these conditions exemplify *special cases*; the Knowledge Math method, like Fu's method before it, fails to satisfactorily interpret a general neural net.

[14] Assuming the test set is truly general and representative of the input domain.

[15] Realistically, the more area the region occupies, the greater the probability the region will catch an input vector. One might argue that increasing the region size will increase the number of vectors that lie within, suggesting a practical upper limit on the complex region size.

How, then, can accurate and useful rules be extracted from a general neural net? A general net interpreter must account for the complex rule regions around the implicit boundary between output classes.

V. Decision Boundaries Abound

Lee and Landgrebe introduced a means of mapping the boundary separating input vectors recognized as belonging to one output class, from those belonging to another[11]. Stewart exploited these *decision boundaries* to determine pertinent features in predicting financial futures[19]. A decision boundary describes a surface through the input space that precisely separates output classes; the activation lines exploited thus far approximate this surface. Being an innate artifact of the neural net, the decision boundary seems rife for symbolic rule exploitation. Perhaps this concept provides a way to extract meaningful relationships in the complex rule regions not covered by the Knowledge Math method.

5.1 What is a Decision Boundary?

A decision boundary denotes a border between two or more output classes. That is, it partitions the input space into independent *classification* or *decision* regions, loosely equivalent to the rule regions found previously. Unlike the activation lines, whose classification boundary is ambiguated by the output sigmoid or tanh function, the decision boundary represents the *exact* inter-class border. The neural net can correctly classify two input vectors adjacent to and on either side of this boundary. Thus, the decision boundary identifies the exact location where the output classification changes from one class to another.

Lee's and Landgrebe's decision boundary provides a means of precisely mapping the decision regions. In terms of symbolic interpretation, the decision boundary connotes a separation of encapsulated rule regions. Can this concept be finessed to discover semiotic relationships within the complex rule regions?

5.2 Finding a Decision Boundary

Consider two input vectors. Suppose the vectors are *correctly classified* by the net into *different classes.* A line drawn between them must cross the decision boundary at some *decision point* (see Figure 5.1). Pairing various combinations of *class 1* and *class 2* points in this manner, one can begin to identify the decision boundary curve (Figure 5.2a).

Ideally, an infinite number of points in one class paired with an infinite number of points in another will enumerate an infinite number of decision points, completely delineating the decision boundary (Figure 5.2b).

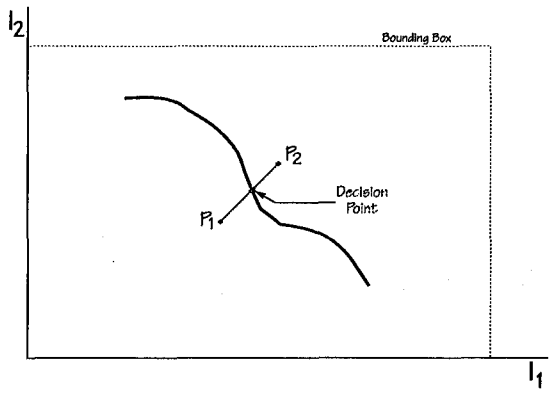


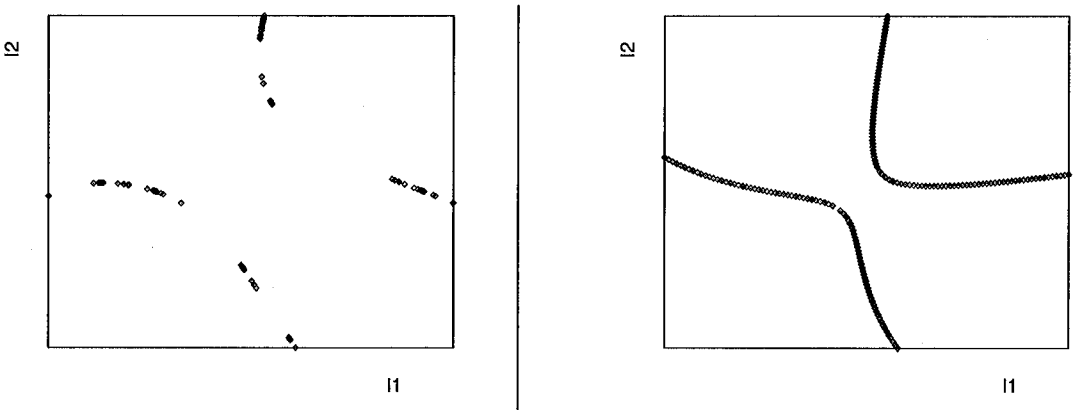
Figure 5.1 **Decision Boundary Point.** Graphically, the decision boundary traces a curve through the input space; a line between opposing points $P_1 \in$ *class 1* and $P_2 \in$ *class 2* intersects the decision curve at a *decision point.*

Of course, the world is not ideal. Slightly less than an infinite number of test vectors will be available for any potential problem domain, implying the decision boundary may not be fully articulated by the set of decision points. Despite these decision boundary *gaps* (Figure 5.2b), the decision points may never-the-less provide enough information to symbolically interpret the complex regions.

Neuron activations do not solely establish the decision boundary because the input space becomes uncertain around the steep sigmoidal (or tanh) midpoint. If the activation line is insufficient, how, exactly, might one find the more precise decision boundary point between two input vectors?

5.3 Discriminant Function

A discriminant function, $h(x)$, can be defined relating input vectors to output classes. In a winner-takes-all output classification strategy, for instance, an input $\overrightarrow{P_i}$ will belong to



(a) Incomplete Boundary **(b)** Ideal Boundary

Figure 5.2 **Articulated Decision Boundary.** A finite amount of training data will lead to an incomplete mapping of the decision boundary by the finite set of decision points (a); invariably, there will be gaps. Ideally, an infinite number of decision points will fully articulate the decision boundary function (b).

class 1 iff $O_1 > O_2$[1]. Similarly, O_2 will be greater than O_1 for $\overrightarrow{P_j} \in$ *class 2*. This suggests a simple discriminant function:

$$h(x) = O_1 - O_2. \tag{5.1}$$

Now, for a *class 1* vector with $O_1 > O_2$, $h(x) > 0$. For a *class 2* vector, $O_1 < O_2$, causing $h(x) < 0$. See Figure 5.3. At some point $\overrightarrow{P_{h_0}}$ between $\overrightarrow{P_i}$ and $\overrightarrow{P_j}$, $O_1 = O_2$ and the discriminant function is 0. Because neither output is larger, neither output class will claim $\overrightarrow{P_{h_0}}$. Therefore, h_0, identifies a single point *between* classes.

Graphically, the I_1-I_2 input plane cuts through "discriminant space" at $h(x) = 0$ as shown in Figure 5.3. Note that all vectors belonging to *class 1* (e.g. P_1) appear on one side of the input plane while all *class 2* points (including P_2) appear on the other. Stepping along the input space line between $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ traces the $h(x)$ curve[2]. At h_0, where $h(x) = 0$, the curve crosses the input plane. In other words, the discriminant function is zero at a

[1]Assume for now that there are only two output classes, *class 1* and *class 2*.

[2]For clarity, the line segment joining P_1 and P_2 in the input space is really the projection of $h(x)$ onto the I_1-I_2 plane.

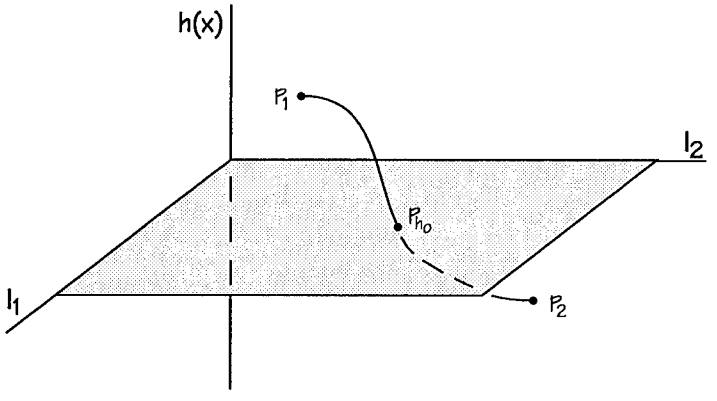


Figure 5.3 **Discriminant Function vs. Inputs.** The discriminant function, $h(x) = O_1 - O_2$, plotted as a function of two input features, I_1 and I_2. At h_0, $h(x)$ passes through zero. This point on the I_1-I_2 input plane, P_{h_0}, represents a point on the decision boundary.

point on the I_1-I_2 plane. The decision boundary point is an actual point in the input space, with input feature values I_{1,h_0} and I_{2,h_0}.

Stepping along the discriminant function, $h(x)$, to find the "zero crossing" can be accomplished by any numerical root finding method[3], such as bracketing, bisection, or the secant method.

This decision boundary more precisely accounts for subsymbolic knowledge embedded within a neural net. Just as Knowledge Math seems a better method than Fu's because it utilizes more aspects of the net knowledge base, the decision boundary could be an even better source of net interpretation. How, then, can it help find rule-type symbolic information? Won't it find complicated, messy rules like those occurring for the Knowledge Math complex rule regions?

5.4 Decision Boundaries, Symbolic Knowledge, and You

The vector-pair line intersects the decision boundary curve at exactly one point, h_0, where the discriminant function passes through 0. Moreover, the *tangent* of the decision

[3]Provided there is only one root, which implies the output classification changes just once.

boundary at this point produces a line or plane that is virtually guaranteed to separate the point, $P_i \in$ *class 1*, from its partner, $P_j \in$ *class 2*.

This condition fails only if both points lie on the tangent line, which is very nearly impossible. If *any* P_i could be coupled with *any* P_j, a meandering decision boundary curve could result in the situation illustrated in Figure 5.4. Pairing two points near a sharp bend in the boundary might, in fact, lead to a tangent line that passes through both points! However, this is an extremely rare case. More importantly, vector pairs can easily be selected ensuring this does not occur. In fact, in a well-trained net with a fairly extensive training set, such pairings can be discarded without general loss of fidelity.

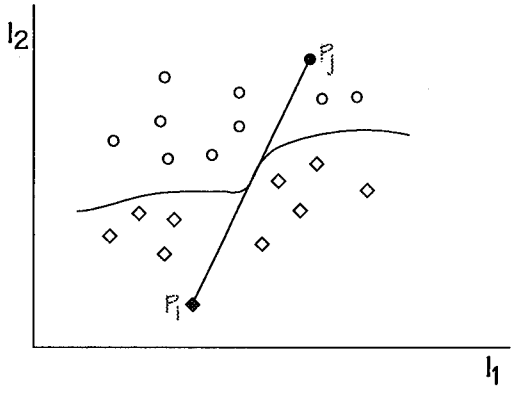


Figure 5.4 **Unfortunate Vector Pairing.** When point P_i pairs with P_j, the resulting vector-pair line intersects the decision boundary as shown. This line also corresponds to the tangent to the decision boundary at the decision point, so the tangent line passes through P_i and P_j. Thus the tangent rules fails to bound P_i from neighbor P_j.

Thus, this tangent represents a *rule* bounding its corresponding point, P_i, in the same manner that activation rules explained input vectors on either side of the activation line in Chapter IV. If every point is paired up with a neighbor in a different class, then every point will have an associated decision point and a tangential rule bounding it to its class. A series of points associated with an encapsulated region will realize of set of rules totally bounding this section from any other. Each rule will be a linear relationship

between inputs. For example, the tangent rule governing P_i will have the form

$$A_{P_i} I_1 + B_{P_i} I_2 + C_{P_i} < 0 \rightarrow \overrightarrow{P_i} \in \textit{class 1}. \tag{5.2}$$

Since a group of such lines together form a *rule set* describing an output class *decision region*, the need to distinguish complex regions from shortcuts disappears. Essentially, the rule set contains a series of *line segments*, bounded where one tangent rule line intersects a pair of others (Figure 5.5).

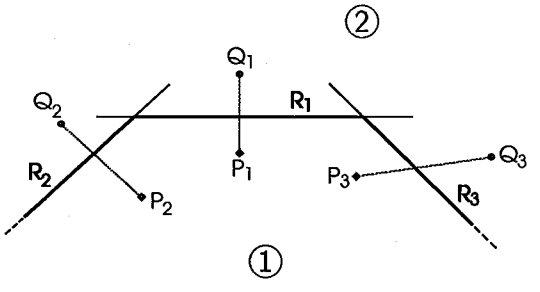


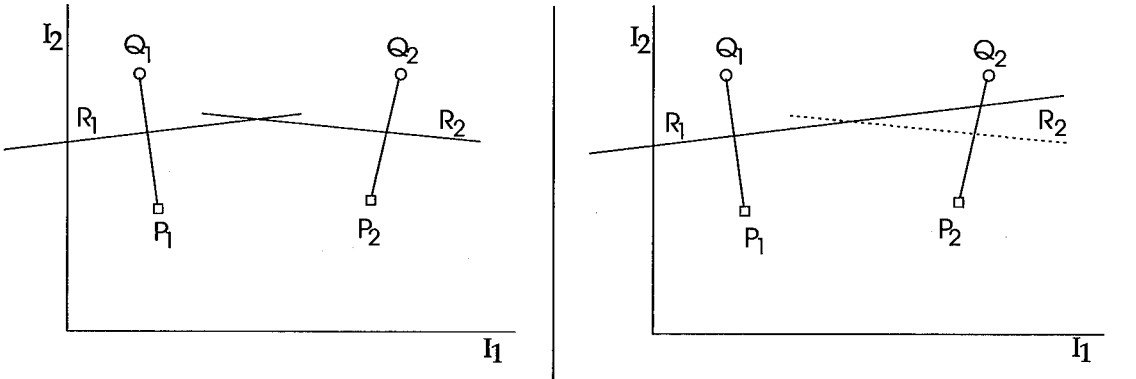
Figure 5.5 **Line Segment Rules.** Tangent rule R_1 separates P_1 from Q_1. This rule can be a member of the rule set bounding Region ① ; in this case, it is valid only between the first intersection on either side by another tangent rule (R_2 on the "left" and R_3 on the "right").

Although each rule in the rule set represents a nice, linear relationship of the inputs—a beneficial characteristic of the Knowledge Math shortcut regions—a decision region contains an arbitrarily large number of training vectors. If there are N_1 points in Region ① , then *at least* N_1 rules govern the decision region[4].

Too many rule relationships restrict the viability of the complete rule set. However, several tangent rules may be very similar. In fact, one rule may sufficiently approximate two or more of them. This is key to the decision boundary's knowledge interpretation role. One point's rule may bound another point. Consider the two points with their assigned neighbors in Figure 5.6a. Although points P_1 and P_2 have unique bounding rules (R_1 and R_2, respectively), the rules are virtually identical. Furthermore, either rule sufficiently

[4]If vectors are allowed to pair with more than one neighbor, more than N_i rules per region could result.

bounds *both* points, as demonstrated in Figure 5.6b. R_1 not only separates P_1 from its neighbor, Q_1, but it isolates P_2 from Q_2 as well. Similarly, R_2 separates P_1 and P_2 from Q_1 and Q_2. One of these rules may be essential (either R_1 or R_2); the other is superfluous and can be discarded from the decision region rule set.



(a) R_1 separates P_1 from Q_1, and R_2 separates P_2 from R_2.

(b) R_2 can be discarded; R_1 sufficiently bounds P_1 and P_2.

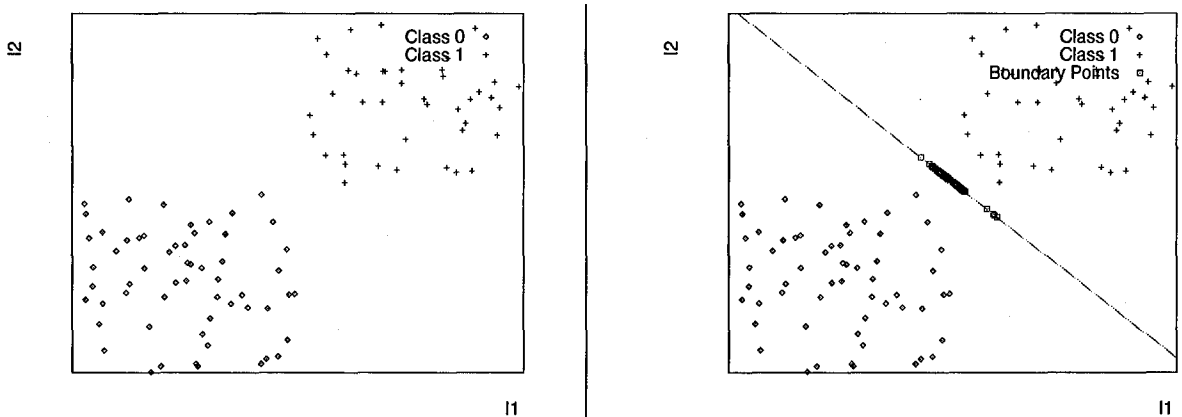
Figure 5.6 **Superfluous Rule Elimination.** R_1 and R_2 separate both P_1 and P_2 from their respective neighbors, Q_1 and Q_2. Thus, one of the rules (R_2, in (b)) can be discarded from the bounding rule set.

Conceivably, a single rule might bound any number of other points in the same class. Because of this, perceived gaps in the decision boundary are not crippling. Articulated points on either side of the gap may sufficiently approximate missing rules associated with potential decision points within the gaps, provided the training vectors are fairly well distributed throughout the input space and the gaps are not terribly large[5]. The goal, then, is to find some subset of tangent rules that sufficiently bounds a decision region.

5.4.1 Linearly Separable Data. For an illustrative example, consider the linearly separable data from Chapter IV (shown again in Figure 5.7a). Pairing every *class 1* point with a *class 2* neighbor (and vice versa) results in a finite series of points approximating the decision curve (Figure 5.7b). However, as inferred from the graph, the individual tangents

[5]Large gaps around "bends" in the decision curve could reduce the precision of the bounding rule set.

combine to form a single straight line suggested by the points themselves. Hence, as expected in linearly separable data, a single rule bounds this domain. It is not coincidental that this decision boundary rule closely mimics the activation line in the Knowledge Math model. As pointed out in Section 4.2, a simple perceptron will sufficiently learn this data, and its single activation line correctly interprets the net knowledge. In other words, the activation line is, in fact, essentially the decision boundary for linearly separable data.



(a) Linearly Separable Data **(b)** Articulated Decision Boundary

Figure 5.7 **Linearly Separable Decision Boundary.** Pairing every $P_i \in$ *class 1* with a *class 2* neighbor and every $P_j \in$ *class 2* with a *class 1* neighbor results in a series of *decision boundary points*, as shown in (b).

Is it this easy to find a set of bounding rules in a general case? Well, no. A general decision boundary inscribes a *curve* through the input space. Within an encapsulated decision region, individual tangent rules can be slightly—or even significantly—different. General rule sets can be found in the following manner.

5.4.2 General Rule Derivation. First, consider the task of pairing vectors. Intuition suggests that pairing $\overrightarrow{P_i} \in$ *class 1* with every vector in *class 2* (statistically, about $\frac{1}{2}$ the test vectors) will produce as complete a mapping of the decision boundary as possible. However, too many vector pairings can effect an exorbitant number of tangent rules, an unnecessarily complex and ugly semantic description, and user head implosion.

Since tangent rules can cover holes in the decision boundary map, it may be more logical to pair vectors as *infrequently* as possible. All vectors must be paired at least once; this ensures that all training cases are verifiably accounted for. A point P_i may or may not be covered by some rule in the decision region rule set. It is difficult to be certain unless P_i has some neighbor to be separated from. Is it sufficient to pair a vector *exactly* once?

Assuming two output classes, $\overrightarrow{P_i} \in$ *class 1* can pair up with any *correctly classified class 2* vector. Joining $\overrightarrow{P_i}$ with its nearest *class 2* neighbor, $\overrightarrow{Q_j}$, virtually eliminates the possibility of both points lying on the tangent line. For a well-distributed training set, nearest neighbor $\overrightarrow{Q_j}$ will be very close to the boundary curve; the continuous curve will likely not bend sharply enough to cause this condition. Moreover, choosing a *nearest* neighbor suggests the tangent line will be almost perpendicular to the vector-pair line. Because the two lines are their most perpendicular, $\overrightarrow{P_1}$ is bounded as well as it can be. Likewise, all points in an encapsulated rule region will be as well-bound as possible. Thus, some sufficient subset of these will nicely and efficiently bound the entire region.

The general decision boundary net interpretation process is accomplished as follows.

5.4.3 ***Step 1:*** *Training.* Train the net normally. The semantic interpretation goal remains: to interpret a *general* net (or, at least, a general multi-layer perceptron). Therefore, neither restrictive architecture constraints nor special learning rules need be applied. The goal requires, first and foremost, a *working* neural net that can be coaxed into explaining itself. Standard backpropagation will work as well as anything.

5.4.4 ***Step 2:*** *Partitioning.* Partition the training set according to output classification. After training is complete (i.e. the error is sufficiently small), split *correctly classified* input vectors into separate output class sets. Unless the net is 100% accurate, some vectors will be misclassified. That is, P_i known to belong to *class 1* may be determined by the net to be in *class 2*. This misclassification implies P_i exists on the wrong side of the border. Is it possible to pair P_i with another vector?

Since P_i is really a *class 1* vector, it *should* be paired with a partner from *class 2*. However, P_i's nearest *class 2* neighbor, Q_j, lies on the same side of the boundary as P_i.

The discriminant function, $h(x)$, between P_i and Q_j will not cross the $h = 0$ input plane. The vector pair line will not intersect the decision boundary between the vectors; indeed, it may not intersect the boundary at all. Thus, no decision point can be ascertained.

Another option is to accept the net's contention and pair P_i with some nearest Q_j belonging to *class 1*. While the vector-pair line *will* cross the decision boundary between the points, the resulting tangent rule does not correctly bound P_i. For instance, a rule bounding misfit P_1 in Figure 5.8 will have the form

$$I_2 \; > \; m_{P_1,\tan} I_1 + b_{P_1,\tan}. \tag{5.3}$$

Clearly, this rule bounds *class 2* vectors, while the opposite rule

$$I_2 \; < \; m_{P_1,\tan} I_1 + b_{P_1,\tan}. \tag{5.4}$$

bounds neighbor Q_1 (and all other *class 1* points). This kind of coupling may identify a unique decision point whose value is limited. A better alternative is to ignore the vector completely. Assuming the training data is expansive enough, the exclusion of misclassified vectors will not adversely affect the completeness of the decision boundary map, or the ability to generate complete rule sets encompassing each decision region.

5.4.5 ***Step 3**: Pairing.* Finding point P_i's nearest neighbor requires calculating the Euclidean distance to *all points* in *every other class.* From the vector difference

$$\begin{aligned} \overrightarrow{P_iQ_j} &= \overrightarrow{Q_j} - \overrightarrow{P_i} \\ &= (Q_{j,I_1} - P_{i,I_1}, Q_{j,I_2} - P_{i,I_2}, \ldots, Q_{j,I_n} - P_{i,I_n}), \quad n = \#\text{ input features} \end{aligned} \tag{5.5}$$

the distance from P_i to Q_j can be found:

$$\begin{aligned} \text{distance} &= \left|\overrightarrow{P_iQ_j}\right| \\ &= \sqrt{(P_iQ_j)_{I_1}^2 + (P_iQ_j)_{I_2}^2 + \ldots + (P_iQ_j)_{I_n}^2} \end{aligned} \tag{5.6}$$

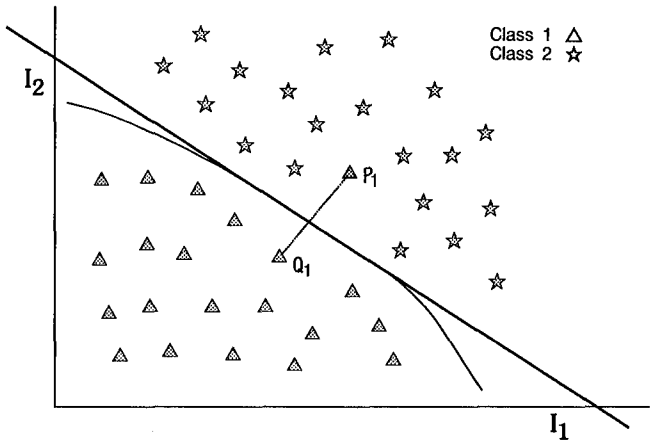


Figure 5.8 **Misclassified Point Bounding.** P_1, a known *class 1* point, falls on the wrong side of the decision boundary. Pairing P_1 with a properly classified *class 1* point, Q_1, finds a decision point and a tangent rule. However, the rule line incorrectly bounds P_1 to the *class 2* side of the decision region.

The closest Q_j becomes P_i's nearest neighbor; the line between them will intersect the decision boundary. The tangent constitutes a *rule line* (or *rule plane*) bounding P_i from Q_j.

5.4.6 ***Step 4**: Bound points.* The decision point falls somewhere on the vector-pair line between P_i and Q_j; the tangent to the decision boundary at this point represents P_i's bounding rule. Every point within a decision region has an associated decision point and, hence, its own rule.

In the extreme, every input vector could realize a *unique* rule. Assuming the training data spans the breadth of the problem domain, this could be sufficient to interpret any potential input vector. Consider a new vector $\overrightarrow{P_{\text{new}}}$ determined by the net to belong to *class 1*. In this case, symbolic interpretation involves finding the closest same-class training vector, $\overrightarrow{P_i}$. If the training data is reasonably dense, the training points will be close together, implying P_{new} will be very near some P_i. That is, P_i's rule will separate P_{new} from the nearest *class 2* point, bounding P_{new} to the appropriate decision region. However, this effects a large number, N_T, of independent rules (N_T = the number of training vectors).

Each μrule is applicable to a very small segment of the input space around its associated training point. Furthermore, finding P_{new}'s closest relative requires computing the distance to every same-class training point. Performing the necessary Euclidean distance calculations for every input vector is expensive.

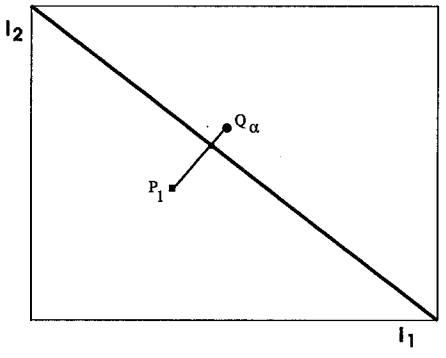
The net-interpretation goal is to find some *sufficient* set of rules bounding vectors within a decision region from all vectors without. Combining and/or discarding individual rules could lead to a subset of these N_T rules encapsulating the entire rule region. The resulting rule subset is sufficient if all of the points within the region are bounded (separated from its nearest neighbor) by *some* rule[6]. That is, every vector should ultimately belong to some rule space *partition* as well.

5.4.6.1 Partitioning. There are two types of rule space partitions applicable to the bounding process: *rule partitions* and *link partitions.* These entities and selected operations on them represent bounding *primitives* necessary to construct an algorithm to find a rule set bounding all the points in a decision region.

Rule Partition. A rule partition identifies a point (called the *ruler* of the partition) whose tangent rule bounds every point in that partition. Typically, a new rule partition, Π_ρ, is created when no existing partition's rule adequately separates P_i from its nearest neighbor Q_j. Thus, P_i is the first point inserted in Π_ρ; P_i becomes the ruler of Π_ρ and P_i's rule becomes the partition rule. Thereafter, another vector P_k that can be bound from its nearest neighbor by P_i's rule may be placed in partition Π_ρ; P_k's individual rule is discarded.

In simple problem domains, rule partitions could be enough to bound all decision regions. For instance, consider the linearly separable data and the resultant decision boundary in Figure 5.4.1. Pick a point, P_1, in *class 1* and find its nearest neighbor, Q_α. P_1 establishes a rule partition, Π_1, whose tangent rule, in fact, describes the decision boundary (see Figure 5.9).

[6]Also, no point from a different class can be contained within the bounded region.



(a) P_1 pairs with Q_α

Bounding Partitions	
Partition	Π_1
Rule	$I_2 < 1 - I_1$
Ruler	P_1
Members	P_1

(b) Resulting Partition

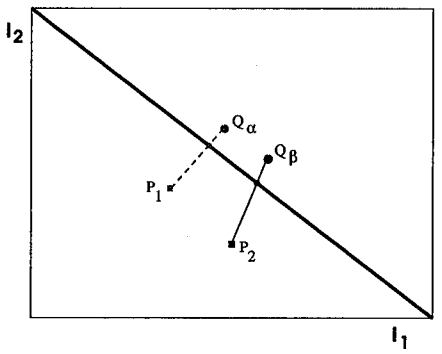
Figure 5.9 **Linearly Separable Data: Bounding First Point.** Pairing P_1 with its nearest neighbor and finding the intersection of this vector-pair line with the decision boundary results in the tangent rule shown in (a). Since this is the first point bounded, a new rule partition is created with P_1 as the ruler (b).

Next, pair another *class 1* vector, P_2, with its nearest neighbor, Q_β. Π_1's rule (i.e. P_1's rule) separates P_2 and Q_β; therefore, P_2 can be added to Π_1, leaving only one partition, as shown in Figure 5.10. In fact, all subsequent *class 1* points can be included in the Π_1, resulting in one symbolic rule describing all of *class 1*. Similarly, a single rule will classify all of *class 2*. In fact, except for some small error, ϵ, inherent in the root finding algorithm, these two rule lines are *exactly the same*, and the net can be summarized by the simple rule set:

$$\begin{aligned} I_1 < \tfrac{B_{p_1}}{A_{p_1}} I_2 + \tfrac{C_{p_1}}{A_{p_1}} \quad &\rightarrow \quad \neg O_1 \\ I_1 > \tfrac{B_{p_1}}{A_{p_1}} I_2 + \tfrac{C_{p_1}}{A_{p_1}} \quad &\rightarrow \quad O_1 \end{aligned} \tag{5.7}$$

However, in more complex problem domains whose decision boundary curves are more complicated, independent rule partitions are not enough.

Partition Link. Consider the XOR data distribution and the resulting decision boundary in Figure 5.11. Since a single line is not sufficient to separate *class 0* from *class 1*, *at least two rules* are necessary to bound any of the four decision regions.

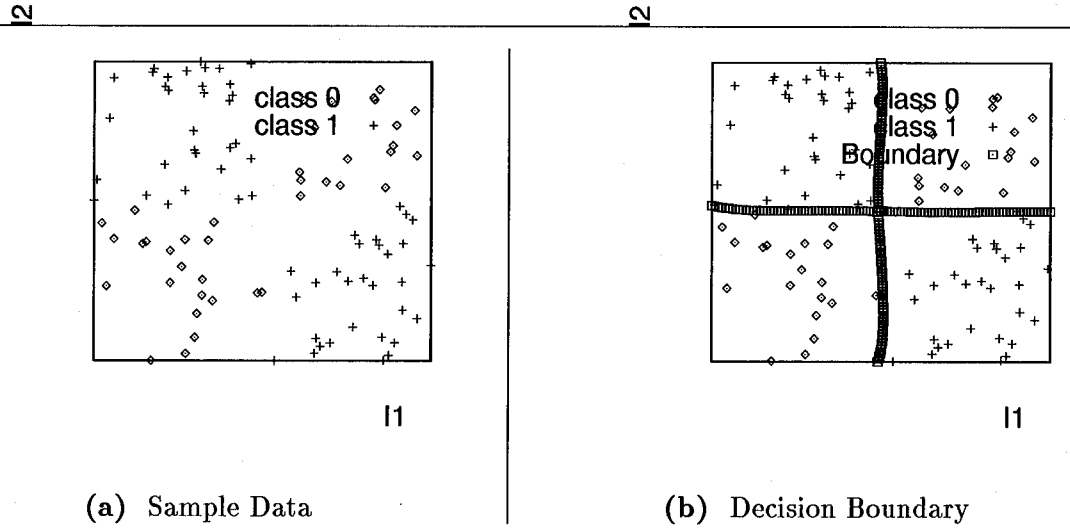


(a) P_2 pairs with Q_β

Bounding Partitions	
Partition	Π_1
Rule	$I_2 < 1 - I_1$
Ruler	P_1
Members	P_1, P_2

(b) Resulting Partition

Figure 5.10 **Linearly Separable Data: Bounding Second Point.** In (a), P_2 paired with nearest neighbor Q_β results in the same tangent rule as P_1. Thus, Π_1's rule bounds P_2 as well as P_1. Hence, P_2 is placed into Π_1; a new partition is not created.



(a) Sample Data

(b) Decision Boundary

Figure 5.11 **Sample XOR Data.** Pairing *class 0* points with *class 1* nearest and *class 1* points with *class 0* neighbors, such as those in (a), produces the set of decision boundary points described in (b).

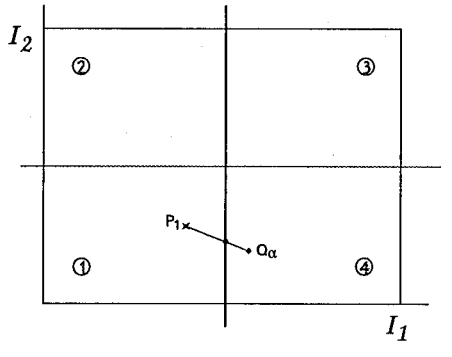
In Figure 5.12, the first point is bounded. $P_1 \in$ *class 0*, in Region ①, is paired up with its closest neighbor, $Q_\alpha \in$ *class 1* (in Region ④). Point P_1 is *ideally* bounded by the rule[7]:

$$I_1 < 0.5 \rightarrow P_1 \in \textit{class 0}, \tag{5.8}$$

or

$$\textbf{IF } I_1 < 0.5 \textbf{ THEN } \neg O. \tag{5.9}$$

Thus, a new rule partition Π_1 is created for P_1.



(a) P_1 pairs with Q_α

Bounding Partitions	
Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1

(b) Resulting Partition

Figure 5.12 **XOR Data: Bounding First Point.** P_1 pairs with Q_α in Region ④, resulting in the "vertical" tangent rule shown in (a). A new rule partition, Π_1, is created to bound this initial point in (b).

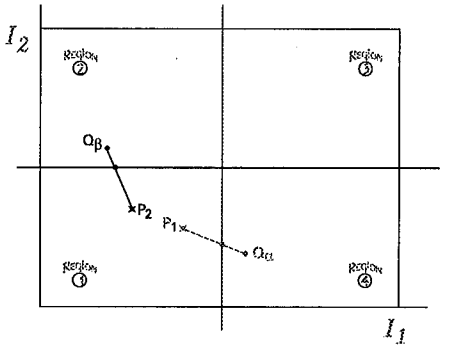
Now consider P_2, whose nearest neighbor Q_β is a *class 1* point in Region ② (Figure 5.13). P_2 is not separated from Q_β by Rule 5.8; Π_1 does not bound this point. A new partition, Π_2, must be created for P_2, with the ideal tangent/partition rule:

$$I_2 < 0.5 \rightarrow P_2 \in \textit{class 0}, \tag{5.10}$$

or

$$\textbf{IF } I_2 < 0.5 \textbf{ THEN } \neg O. \tag{5.11}$$

[7]The actual rule line may "lean" somewhat; the accuracy of the neural net approximation depends on the completeness of the training data.



(a) P_2 pairs with Q_β

Bounding Partitions	
Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2

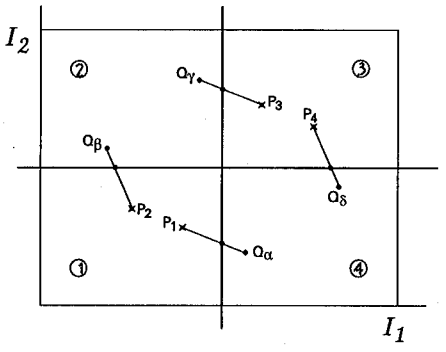
(b) Resulting Partitions

Figure 5.13 **XOR Data: Bounding Second Point.** P_2 pairs with Q_β in Region ②. The resulting "horizontal" tangent rule is shown in (a). Π_1's rule does not separate P_2 from Q_β; hence, a new rule partition is created for P_2, as shown in (b).

Notice in Figure 5.13 that the rules of Π_1 and Π_2 describe different sides of the box containing Region ①. Clearly, P_1 and P_2 lie in the same encapsulated decision region. However, if the partitions remain independent, Π_1's member set will be recognized as disjoint from that of Π_2.

Thus it becomes necessary to introduce the concept of a *link* within a partition. Π_1's rule does not separate P_2 from its neighbor, Q_β; nor does Π_2's rule separate P_1 from Q_α. However, Π_1's rule *does* separate P_2 from $\boldsymbol{P_1}$'s neighbor, just as Π_2's rule separates P_1 from $\boldsymbol{P_2}$'s neighbor. A dependency exists between Π_1 and Π_2. partitions 1 and 2.

To handle this dependency, a *link set* is added to the rule partition structure; between every pair of rule partitions that meet the dependency criteria, a link is established. Figure 5.14 illustrates partition links, such as the one between Π_1 and Π_2. Notice that the eight rule partitions describing pertinent domain rules are no longer independent; pairs of partitions are linked, resulting in four rule sets bounding the four XOR decision regions (one set per region). Table 5.1 summarizes the resulting rules.



(a) P_1, P_2, P_3, and P_4 are mutually paired with Q_α, Q_β, Q_γ, and Q_δ, bounding all four regions[9].

Bounding Partitions

Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Links	Π_2

Partition	Π_3
Rule	$I_1 > 0.5$
Ruler	P_3
Members	P_3
Links	Π_4

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Partition	Π_4
Rule	$I_2 > 0.5$
Ruler	P_4
Members	P_4
Links	Π_3

Partition	Π_5
Rule	$I_1 > 0.5$
Ruler	Q_α
Members	Q_α
Links	Π_6

Partition	Π_7
Rule	$I_2 > 0.5$
Ruler	Q_β
Members	Q_β
Links	Π_8

Partition	Π_6
Rule	$I_2 < 0.5$
Ruler	Q_δ
Members	Q_δ
Links	Π_5

Partition	Π_8
Rule	$I_1 < 0.5$
Ruler	Q_γ
Members	Q_γ
Links	Π_7

(b) Entire Partition Set

Figure 5.14 **XOR Data: Complete Bounding.** Two points in each region paired with neighbors as shown in (a) results in the linked partitions in (b).

R1	**IF** $(I_1 < 0.5) \wedge (I_2 < 0.5)$	**THEN** $O \in$ *class 0*
R2	**IF** $(I_1 > 0.5) \wedge (I_2 > 0.5)$	**THEN** $O \in$ *class 0*
R3	**IF** $(I_1 > 0.5) \wedge (I_2 < 0.5)$	**THEN** $O \in$ *class 1*
R4	**IF** $(I_1 < 0.5) \wedge (I_2 > 0.5)$	**THEN** $O \in$ *class 1*

Table 5.1 **Decision Boundary XOR Rule Set.** The four pairs of linked rule partitions in Figure 5.14 result in this set of rules governing the XOR domain.

Thus, linearly separable and XOR domains are partition adequately with rule partitions. However, even incorporating inter-partition links, rule partitions are not necessarily enough to correctly interpret decision regions.

Link Partition. Consider the Stair-Step problem. Although the input space is divided into only two decision regions, the decision boundary is much more complicated than those of linearly separable or XOR problems. Notice the single step around P_1, P_2, and P_3. This section of Region ① appears very much like one of the XOR quadrants; it behaves similarly as well. That is, P_1 and P_2 will be placed into separate rule partitions (Π_1 and Π_2) with a link between them. P_3 adds no new information and can be bound by either partition. P_3 might be placed in Π_1; P_3's rule is discarded.

Likewise, P_a and P_b, on another step, are placed into separate, linked partitions Π_3 and Π_4), respectively. Either Π_3 or Π_4 will contain P_c.

According to the bounding mechanisms discussed thus far, these two sections (i.e. two partition sets, $\{\Pi_1,\Pi_2\}$ and $\{\Pi_3,\Pi_4\}$) will remain independent of each other. There exists no established way to link these two sets together. Note, though, that P_χ can be bounded by both Π_1 and Π_3, suggesting a link should exist between these rule sets, as has already been intuited. However, inserting P_χ in either Π_1 or Π_3 discards all knowledge about P_χ, including this necessary link. Thus a new type of entity is required: a *link* partition.

A link partition behaves much like a rule partition. However, it does not include a *rule* or *ruler*. It contains only a set of member points and a set of links to those *rule partitions* that can bound the points contained within. Thus, before P_χ is bounded to a partition, two independent rule sets exist (see Table 5.2). P_χ elicits the creation of a new *link* partition, Π_5, with links to Π_1 and Π_3. After bounding P_χ, all the partitions are linked (Table 5.3).

Now, an association can be traced between the old rule sets, $\{\Pi_1,\Pi_2\}$ and $\{\Pi_3,\Pi_4\}$), enabling their conflation into a single set $\{\Pi_1,\Pi_2,\Pi_3,\Pi_4\}$). Note that Π_5 exists only to link the other partitions together. It contains no rule information; hence, Π_5 is not a member of the bounding rule set.

Bounding Partitions

Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Links	Π_2

Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Links	Π_2

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Table 5.2 **Stair-Step Rule Partitions.** Bounding points P_1, P_2, P_3, P_a, P_b, and P_c in A simple stair-step problem results in two independent sets of partitions: Π_1, Π_2 and Π_3, Π_4.

Bounding Partitions

Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Links	Π_2

Partition	Π_1
Rule	$I_1 < 0.5$
Ruler	P_1
Members	P_1
Links	Π_2

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Partition	Π_2
Rule	$I_2 < 0.5$
Ruler	P_2
Members	P_2
Links	Π_1

Table 5.3 **Stair Step with Link Partition.** Recognizing that P_χ can be bounded by independent rule partitions Π_1 and Π_3, P_χ is placed into a *link* partition. As a result, all of the rule partitions are linked together.

Still, the concept of partitions is not complete. Utilizing rule and link partitions, and linking and bounding points as described thus far, an arbitrary decision region may *still* not be bounded correctly.

Link Breaking. Consider the Island Problem in Figure 3.3.3. (include points 1,2,3,4). In this problem, two independent *class 0* "islands" are completely surrounded by a *class 1* "sea".

Initially, P_1 is placed into its own rule partition, Π_1. Although P_2 requires a new rule partition, Π_2, Π_1's rule separates P_2 from Π_1's ruler's (P_1's) neighbor. Also, Π_2 separates P_1 from P_2's neighbor. Therefore, a link is established between Π_1 and Π_2, just as before. However, Π_1 and Π_2 should *not* be linked in this case. This problem must be resolved.

Now bound P_3. Since neither Π_1 nor Π_2 separates P_3 from its neighbor, P_3 requires a new rule partition, Π_3. Π_3 can now be linked to Π_1, but not Π_2. That is, Π_3 can be linked to only one element of the rule set $\{\Pi_1,\Pi_2\}$. Since Π_1 is already linked to Π_2, an inconsistency exists. Breaking the Π_1-Π_2 link and recompeting all three partitions for applicable links can solve this problem. Before any link is established (or reestablished), consistency between Π_1, Π_2, and Π_3 must be maintained.

Thus, the ability to break and recompete links becomes the final element necessary to bound vectors within decision regions. With this set of vector-bounding primitives (see Table 3.3.3), a bounding algorithm can be constructed.

5.4.6.2 Algorithm. Too many rules are simply too much. An algorithm to bound vectors within decision regions could attempt to find an optimal bounding; however, it is unclear exactly what "optimal" means here (see Section 5.5.2 below). To develop an appreciation for this difficulty, start with the basics. The algorithm presented here finds *a* solution. While it may not the *best* solution, it is the *easiest.* More importantly, it provides clues about *how* the bounding problem might be optimized.

, ,

Train net as usual (e.g. with backpropogation)
Partition *correctly classified* test vectors into separate sets, S_k (i.e. one set per output class).
Find nearest neighbors, Q_j, for all vectors, P_i, in each output class set, S_k. , ,

For each vector $\overrightarrow{P_i} \in$ each class set, S_k , ,
Calculate the Euclidean distance to every vector in all S_l, $l \neq k$

Remember nearest neighbor, $\overrightarrow{Q_j}$
Tag $\overrightarrow{Q_j}$ to $\overrightarrow{P_i}$[10]
Partition all points in each set into a *rule* or *link* partition , ,
For each point $P_{k,i}$ in S_k
Is the point already bounded? That is, does it already belong to a partition?
If so, continue with next point
If not , ,
Find all rules that can bound it , ,
Look at each *rule* partition
Does the partition's rule separate $P_{k,i}$ from its neighbor $Q_{l,k}$?
Add up "yes"'s
How many established rules (i.e. existing rule partitions) can bound $P_{k,i}$? , ,
If more than 1, then the point *links* affected rule partitions , ,
$\exists$ a link partition whose links are exactly the same as point's bounders?
If yes, add point to that link partition
If no, create a new link partition, adding all bounders to the new partition's links
If exactly 1, then the point belongs to the appropriate rule partition—insert it
If none, then the point belongs to a new rule partition , ,
create a new rule partition
check other rule partitions for applicable links , ,
Does the other partition's rule separate me from his neighbor?
Does my rule separate the other partition's point(s) from my neighbor?
If yes to both, establish link between them

At this point, every correctly classified training datum is included in some partition, according to the constraints detailed above. The set of partitions describe every decision region; dependent partitions bound individual regions. However, before bounding is complete, a final check must be made to ensure the links are complete and consistent.

5.4.7 ***Step 5****: Transitivity.* In the algorithm, a new rule partition is immediately linked to any other appropriate rule partition. However, transitivity is not enforced at the time. Obviously, if Π_A is linked to Π_B and Π_B is linked to Π_C, then Π_A should be linked to Π_C as well.

In particular, link partitions contain rule partitions that should be linked but have not been. This is not enforced during bounding because of the potential need to severe and reestablish links.

[10] Q_j is P_i's nearest neighbor, but P_i may or may not be Q_j's.

After bounding is complete, transitivity is accomplished by stepping through the partitions ensuring link consistency. Now, independent rule sets, bounding encapsulated decision regions, can be readily ascertained. However, the rule sets may be laden by lots of rules; ideally, superfluous rules should be culled out.

5.4.8 ***Step 6:*** *Subsuming.* Subsumption represents an obvious optimization. In the simple algorithm presented here, order matters. For instance, the order that points were partitioned in the Island Problem constituted the need to break and recompete links. Picking the points in a special order (say, P_1, then P_3, followed by P_2) could have avoided this situation.

In general, the order in which points are bounded can impact which—and how many—rules are contained within the final rule sets. Arguably, small is better. That is, fewer tangent rules typically provide more semantically acceptable net explanations. In an XOR problem, P_1 and P_2 can be partitioned as before, resulting in partitions Π_1 and Π_2 which totally describe Region ① with the rule:

$$I_1 < 0.5 \wedge I_2 < 0.5 \rightarrow O \in \textit{class 0}. \tag{5.12}$$

P_3, in the "corner" of Region ①, can then be added to either partition or produce its own link partition. On the other hand, Figure 3.3.3b shows the same domain, but the points are bounded in reverse order. Now, P_3 is bound first. Without any existing partitions to bound the point, a new rule partition must be created (Π_1). However, Π_1 does not bound P_2 or P_1. Separate rule partitions are needed for each of these points, resulting in the rule:

$$I_1 < 0.5 \wedge I_2 < 0.5 \wedge I_1 < m_{P_1} I_2 + b_{P_1} \rightarrow O \in \textit{class 0}. \tag{5.13}$$

Clearly, Rule 5.12 is more palatable. More importantly, rule partition Π_1 is superfluous in the latter case and can be *subsumed.* As a post-processing step, find and remove those partitions such as Π_1 whose ruler can be separated from its neighbor by another rule in the rule set. Continue this until the rule set can be reduced no further.

After all training vectors have been bound, the transitivity check has occurred, and rules have been subsumed as appropriate, the augmented net is ready for general use. All decision regions have been described by a set of rules. Any vector applied to the net during normal use will be classified and, if requested, an explanation (based on the rule set) can be provided to the user.

5.5 Results

Use the net. All the rule generation pre-processing is complete. Give it a vector. Ask for an explanation. The net will classify the input. If asked to supply an explanation, the expert network will further determine in which decision region the vector lies. This region has a unique rule set associated with it. The net supplies these *rule relationships* to the user.

Note that the input vector can be misclassified[11]. In this case, incorrect rules will be given to the user. If the net believes a vector $\overrightarrow{P_i}$ belongs to *class 1*, for instance, the rule set describing the appropriate *class 1* region will be returned. That is, the explanation facility will be no more accurate than the net.

In some cases, a misclassified vector's rules may not make sense, and the user may be able to determine that a misclassification has occurred. In other cases, where the user cannot distinguish good rules from bad, the user must rely on the statistical accuracy of the net to accept the solution and explanation. However, humans are typically wary of computers; it would be preferable that the net and, hence, the decision region rule set *always* correctly explained every input.

Actually, the nature of the rule reduction presented here is such that the governing rules are only an approximation of the rule region. The fewer number of rules in a rule set, the less precisely bound the decision region. Is this good enough? Remember that the net itself learns only an approximation of the input domain; a more expansive training set will result in a better trained net. However, there is always the potential, for any problem

[11] This is an artifact of an imperfectly trained net, or a lack of generalization in the training data, *not* a result of the rule generation process.

domain, that an obscure input vector will invariably be misclassified, no matter how well or how general the net is trained. This risk is assumed by artificial neural net users.

Similarly, the rule sets approximate the inherent decision regions of the net. Thus, a rule set is typically an approximation of an approximation. Like the net itself, the rule sets can bound the decision region tightly, catching virtually every potential input vector. Still, it is conceivable that a vector may get classified correctly by the net, while not being covered by any rule.

For example, consider a bounding rule, R_1, slightly offset from the "ideal" rule, $I_1 < 0.5$. Such a rule could have been chosen because of the order in which the points were bounded, or perhaps the net has not learned the domain very strictly. In either case, P_α might be classified *correctly.* That is, the net will claim P_α is a *class 1* point. However, when searching for an explanation, P_α is *class 1* but lies technically within a *class 2* decision region. What can the net do? It can: (1) return the wrong answer (the *class 2* rule set), (2) provide an error message in lieu of a rule set, or (3) find the closest *class 1* decision region. The first two should be avoided; the latter is easily accomplished (though not necessarily quickly) by finding the nearest *class 1* point P_i and returning P_i's bounding rules. This is quite clean; uninterpreted points will likely fall very close to a decision region of the appropriate class.

Therefore, the Decision Boundary net-interpretation method generates decent symbolic rule-relationships in the complex regions not covered by Fu or Knowledge Math. Along with Knowledge Math, it seems to provide a means of fully interpreting a general neural net. In fact, this method alone produces rule sets bounding every input vector. It would seem, then, that the decision boundaries provide the explanation facility desired in a neural net. However, this approach suffers from limitations as well.

5.5.1 Limitations. First, the rule sets may be heavily laden with rules. Every training vector within a decision region can realize a unique rule. In the extreme case, the number of rules in the rule set will be the same as the number of training vectors enclosed within a decision region (i.e. the rule set contains one rule per vector). Since the number of vectors bounded within a given region can be arbitrarily large, the rule set can

easily become semantically unwieldy. Multiple rules corresponding to rule relationships that cover very small regions can be cumbersome and ugly.

The bounding algorithm addresses this by providing a mechanism to subsume superfluous rules. As a result, the size of the rule set is reduced as much as possible. Although this attempts to reduce the impact of one limitation, it brings up another.

The resultant rule subset may not tightly approximate the decision boundary curve. As the approximation becomes looser, the chance of finding an uninterpretable vector (i.e. P_i is classified as *class 1* but falls outside the *class 1* bounding rules) increases. As was suggested previously, P_i can be adequately explained by finding its closest relative, P_j, within a *class 1* decision region. As long as P_i is close to P_j, P_j's rule set will sufficiently cover P_i as well.

However, as the rule set approximation of the decision boundary becomes less exact, more vectors can lie in these uninterpretable areas. Finding P_i's closest relative requires calculating the distance to every known *class 1* point (or, at least, one such point in each encapsulated rule region). The distance calculations are expensive; to perform this frequently, the interpreter will take a performance hit.

Finally, order affects how well decision regions are bounded and how many rules are required to adequately bound them.

As has already been belabored, order matters. The order in which points are bounded within the bounding algorithm Despite the subsumption provided after the bounding algorithm, which itself is an expensive exhaustive search, the algorithm provides no guarantee that a "best" rule set is found. Indeed, it fail to define what a "best" rule set is! Even in subsumption, order matters. In particular, the case of the ill-interpreted vector can be made rare by subsuming in the right sequence (say, getting rid of the rule shown in Figure 3.3.3 in favor of a more "vertical" one).

The other limitations can be readily handled; this one deserves a closer look.

5.5.2 A Note on Optimization. The concept of a set of "best" rules governing a decision region is hard to define. "Best" implies something to be optimized, but what?

The number of rules? Ultimately, the definition of "best" must be the most semantically acceptable to the user. But what is acceptable by the user? This may well be domain specific. An engineering or scientific problem—or anywhere where precision is paramount—a *maximum* number of rules or the tightest, most complex bounding of the decision region may be in order. In many control applications, such as a virtual copilot, explanations should be as easy (i.e. *fewest* rules) as possible. Stock market prediction, where the goal is to guess whether the market will rise or fall, but where implicit relations between indices are not easily determined, probably requires something in between.

Therefore, an optimization function is probably best supplied with *a priori* domain knowledge. There are three basic forms such optimization could take: *quickest*, *fewest*, and *tightest.*

The *quickest* rule set was more or less what was found by the algorithm presented here. This is probably sufficient for many general problem domains. Certainly it works well for toy problems, such as linearly separable or XOR data. But the choice of points to bound is random; thus the method as a whole is ad hoc. There is no guarantee that the resulting rules will most efficiently describe the enclosed regions. The size of the rule set may be reduced by subsuming superfluous rules. Subsumption will "smallify" the rule set into as semantically nice a set of rules as possible, based on the random ordering of points.

The *fewest* optimization suggests finding the minimum number of rules. Note that this is an exhaustive search of a combinatoric rule space. Basically, all possible orderings of the input vectors must be considered. Being a minimum number of straight-line rules, this could provide even better semantics than the *quickest* algorithm described here. However, such minimalistic rule set bound the decision region as loose as inhumanly possible. In the worst case, this optimization implies finding the rule set that *maximizes* the area bounded by the potential set of tangent rules. Thus the potential of losing outliers is increased. Furthermore, the un-interpreted outliers may not be as so close to a region of the right class, so the approximation necessary to explain the vector by the nearest neighbor's rule set increases.

The final general means of optimization is to find the *tightest* rules. That is, find the "best fit" of the decision region by some subset of tangent rules. Essentially, this rule subset *minimizes* the enclosed input space area. This, too, *could* result in an exhaustive search of the the straight line tangent rules. However, another approach is to find a single, potentially quite complex function to inscribe a very tight approximation of the boundary suggested by the set of decision points. For example, a curve fitting technique such as minimum description length encoding (MDL) could be employed to find an approximating function[12].

In general, *tightest* optimization implies finding few (possibly one) "nice" bounding equation. For instance, consider the island problem. Each island can be approximated very nicely by an ellipse. However, this ellipse equation is not as esthetically pleasing as straight line rules. That is, it is harder to determine the nature of the rule relationship without plotting it graphically. Even then, the rule may be complex enough that all semantic value is lost.

Thus, to recapitulate, the *quickest* optimization, which happens to be the simplest conceptually and implementation-wise, may be the best in the general case. Random rules may be the best trade off between tight, ugly rules and pretty, loose rules.

VI. Conclusions and Recommendations

Fu established a good neural net symbolic interpretation foundation. However, he traded precision and accuracy for an independent rule base that realized only an approximation of the net. Although he claimed the rule base performed comparably with the original net[1], his method failed to address augmentation of the neural net with an explanation facility. In particular, in an ambiguous domain such as stock market prediction, the important information is not just the input features but the *interaction* among them. Human experts may not know what input relations are important; the net could provide rule relations, increasing domain knowledge.

The Knowledge Math mathematical extension of Fu, developed in this work, better addresses net augmentation and rule relationships. This method employs a precise knowledge interpretation, it is intuitively nice, and it provides very good special cases (shortcuts).

The Decision Boundary method, also developed in this research, solves these complex rule regions. Indeed, this method solves the whole implementation problem very well. It is grounded not only on the *precise* knowledge of the decision boundary in the net, which is only approximated by the activation lines used by Fu or Knowledge Math.

The rules generated by the Decision Boundary method may not be as nice as those generated by the Knowledge Math method. In particular, multiple tangent line segment rules bound any given decision region; with these, it may be hard to discern the rules' semantic value. The Decision Boundary method could be augmented by Knowledge Math. Knowledge Math can provide "quick" and "nice" rules for any existing shortcut rule region. As noted in Section 3.3.3, these shortcut rules are, in fact, the most general rules and rule relations possible in the net. These rules are simple to understand. A "quick check" of the input space to determine if any shortcut rules apply could be very beneficial to the user.

If the vector lies within a Knowledge Math complex rule region, the Decision Boundary method provides potentially less nice rules, but it will find essential relationships governing the output classification in that decision region. It is important to note that

[1] In terms of accurate output classification; not in terms of speed or efficiency.

sometimes life, like rule relationships, is complicated. If the user really wants to know why things are happenings as they are for an input that lies within the complex rule region, then he is going to have to face the possibility of complicated rules.

In summary, the Decision Boundary method, or Knowledge Math in conjunction with Decision Boundaries, will provide symbolic knowledge whose usefulness surpasses Fu's, because it better utilizes subsymbolic net knowledge.

Bibliography

1. Caudill, Maureen. "Expert Networks," *BYTE*, 108–116 (October 1991).
2. Cybenko, George. *Approximations by Superpositions of a Sigmoidal Function*. Research Note, Computer Science Department, Tufts University, October 1988.
3. Frasconi, P., et al. "An Unified Approach for Integrating Explicit Knowledge and Learning by Example in Recurrent Networks." *Proceedings of the IJCNN 1*. 811–816. 1991.
4. Fu, Li Min. "Knowledge-Based Connectioinism for Revising Domain Theories," *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(1):173–182 (1993).
5. Fu, Li Min. "Rule Generation from Neural Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(8):1114–1124 (August 1994).
6. Gallant, Stephen I. "Connectionist Expert Systems," *Communications of the ACM*, *31*(2):152–169 (1988).
7. Giarratano, Joseph and Gary Riley. *Expert Systems* (Second Edition). PWS Publishing Company, Boston, Massechusetts, 1994.
8. Handelman, D. A., et al. "Integration of Knowledge-based System and Neural Network Techniques for Autonomous Learning Machines." *Proceedings of the IJCNN 1*. 683–688. 1989.
9. Hruska, S. I., et al. "Hybrid Learning in Expert Networks." *Proceedings of the IJCNN 2*. 117–120. 1991.
10. Kane, Raqui and Maurice Milgram. "Extraction of Semantic Rules from Trained Multilayer Neural Networks." *IEEE International Conference on Neural Networks 3*. 1397–1401. IEEE, New York, NY, 1993.
11. Lee, Chulhee and David A. Landgrebe. "Feature Extraction Based on Decision Boundaries," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(4):388–400 (1993)..
12. Li, Mengxiang. *Minimum Description Length Based 2D Shape Description*. Technical Report, Computational Vision and Active Perception Laboratory (CVAPL), Department of Numerical Analysis and Computing Science, Royal Institue of Technology, Stockholm, Sweden, October 1992.
13. McMillan, C., et al. "Learning Explicit Rules in a Neural Network." *Proceedings of the IJCNN 2*. 83–88. 1991.
14. Mitra, Sushmita and Sankar K. Pal. "Fuzzy Multi-Layer Perceptron, Inferencing and Rule Generation," *IEEE Transactions on Neural Networks*, *6*(1):51–63 (1995).
15. Newell, Allen and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.
16. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*. SPIE Optical Engineering Press, 1991.
17. Rumelhart, D. E., et al. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*. MIT Press, Cambrise, MA, 1986.

18. Samad, T. "Towards Connectionist Rule-based Systems." *Proceedings of the IJCNN 2*. 525–532. 1988.

19. Stewart, James A. *Nonlinear Time Series Analysis*. Master's Thesis, Air Force Institute of Technology, 1995.

20. Tazaki, Eiichiro and Norimasa Inoue. "Automated Extraction of Fuzzy Rules Using Neural Networks with Planar Lattice Architecture." *Moving Towards Expert Systems Globally in the 21st Century* edited by Jay Liebowitz, 1221–1227, Cognizant Communication Corporation, 1994.

21. Towell, G. G., et al. "Refinement of Approximate Domain Theories by Knowledge-based Neural Networks." *Proceedings of AAAI*. 861–866. 1990.

22. Šíma, Jiří. "Neural Expert Systems," *Neural Networks*, *8*(2):261–271 (1995).

23. Winston, Patrick Henry. *Artificial Intelligence* (Thrid Edition). Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

24. Yang, Q. and V. K. Bhargava. "Building Expert Systems by a Modified Perceptron Network with Rule-transfer Algorithms." *Proceedings of the IJCNN 2*. 77–82. 1990.

25. Yeung, Daniel S. and H. S. Fong. "A Knowledge Matrix Representation for a Rule-Mapped Neural Network," *Neurocomputing*, *7*:123–144 (1995).

26. Yeung, S., et al. "A Neocognitron-Based Chinese Character Recognition System." *Proceedings of the IJCNN 3*. 617–622. 1992.

27. Zadeh, Lofti A. "Fuzzy Sets," *Information and Control*, *8*:338–353 (1965).

28. Zadeh, Lofti A. "The Role of Fuzzy Logic and Soft Computing in the Conception and Design of Intelligent Systems." *Fuzzy Logic in Artificial Intelligence*, edited by E. P. Klement and W. Slany. 1993.

Vita

Capt Stanley Dale Kinderknecht [redacted], Kansas. After high school, he attended Kansas State University, where he received Bachelor of Science degrees in Computer Engineering and Electrical Engineering in 1991. Upon receiving a commission through Air Force Reserve Officer Training Corps at Kansas State, he was stationed at Wright-Patterson AFB, Ohio. After three years in Aeronautical Systems Center, he was accepted into the Master's program at the Air Force Institute of Technology in June of 1993.

Stanley married Lisa Ann Bollig in 1987. In eight years together, they have jointly produced six beautiful children: Christopher Stanley, Corwin James, Candace Nicole, Cian Alexander, Collin Joseph, and Cameron Kenneth.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Master's Thesis

4. TITLE AND SUBTITLE

SEMANTIC INTERPRETATION OF AN ARTIFICIAL NEURAL NETWORK

5. FUNDING NUMBERS

6. AUTHOR(S)

Stanley D. Kinderknecht

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/95D-07

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

Recent advances in machine learning theory have opened the door for applications to many difficult problem domains. One area that has achieved great success for stock market analysis/prediction is artificial neural networks. However, knowledge embedded in the neural network is not easily translated into symbolic form. Recent research, exploring the viability of merging artificial neural networks with traditional rule-based expert systems, has achieved limited success. In particular, extracting production (IF...THEN) rules from a trained neural net based on connection weights provides a valid set of rules only when neuron outputs are close to 0 or 1 (e.g. the output sigmoid function is saturated). This thesis presents two new ways to interpret neural network knowledge. The first, called Knowledge Math, extends the use of connection weights, generating rules for general (i.e. non-binary) input and output values. The second method, based on decision boundaries, utilizes the inherent border between output classification regions to draw symbolic interpretation. The Decision Boundary method generates more complex symbolic rules than Knowledge Math, but provides valid feature relationships in the uncertain regions around the midpoints of the neuron output functions. The main result is a complementary relationship between Knowledge Math and Decision Boundaries, as well as subsymbolic and symbolic knowledge representations for a general multi-layer perceptron.

14. SUBJECT TERMS

Neural Networks, Rule-Based Expert Systems, Rule Extraction

15. NUMBER OF PAGES

88

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.